# MULTI-STANDARD SOFTWARE DEFINED RADIO IMPLEMENTATION USING PARTIAL DYNAMIC RECONFIGURATION

By

Abdelrhman Hussien Mohamed Ibrahim

Ali Hesham Mohamed El Shazly

Basma Ashraf Mohamed Mohamed

Islam Mohamed El Nader Mahmoud Hassan

Mohamed Ossama Ashour Mokhtar

Mostafa Gamal Ahmed Mohamed

Under the Supervision of

Associate Prof. Hassan Mostafa

Associate Prof. Yasmine A.H. Fahmy

A Graduation Project Report Submitted to
the Faculty of Engineering at Cairo University
In Partial Fulfillment of the Requirements for the
Degree of Bachelor of Science
in
Electronics and Communications Engineering

Faculty of Engineering, Cairo University

Giza, Egypt

July 2017

# Table of Contents

# List of Tables

# List of Figures

# List of Symbols and Abbreviations

| | |
|---|---|
| 1G | first generation |
| 2G | second-generation |
| 3GPP | 3rd generation partnership project |
| ACP | Accelerator Coherency Port |
| ADC | Analog to Digital Converter |
| AP SoC | All Programmable SoC |
| ASIC | Application Specific Integrated Circuit |
| BPSK | Binary Phase Shift Keying |
| CLBs | Configurable Logic Blocks |
| CP | cyclic prefix |
| CR | Cognitive Radio |
| CRC | Cyclic Redundancy Check |
| DAC | Digital to Analog Converter |
| DCM | Digital Clock Management |
| DCP | Design Check Point |
| DMA | Direct Memory Access |
| DPC | Dirty Paper Coding |
| DPCCH | Dedicated Physical Control Channel |
| DPCCH2 | Dedicated Physical Control Channel 2 |
| DPDCH | Dedicated Physical Data Channel |
| DSP | Digital Signal Processing |
| EDGE | Enhanced Data rates for GSM Evolution |
| E-DPCCH | E-DCH Dedicated Physical Control Channel |
| E-DPDCH | E-DCH Dedicated Physical Data Channel |
| ETSI | European Telecommunications Standards Institute |
| FBI | Feed-Back Information |
| FCC | Federal Communications Commission |
| FF | Flip Flops |
| FPGA | Field Programmable Gate Array |
| FSM | Finite state machine |
| GP | General Purpose |
| GPP | General Purpose Processor |
| GPRS | General Packet Radio Services |
| GSM | Global System for Mobile |
| HDL | Hardware Description Language |

| | |
|---|---|
| HF | High Performance |
| HS-DPCCH | Dedicated Control Channel with HS-DSCH transmission |
| HSPA | High Speed Packet Access |
| IA | Intelligent Antenna |
| IC | Integrated Circuit |
| IEEE | Institute of Electrical and Electronics Engineers |
| LNA | Low Noise Amplifier |
| LTE | Long Term Evolution |
| LUTs | Look up Tables |
| MAC | Medium Access Control |
| MPDUs | MAC Protocol Data Units |
| MSCS | Multi-Standard Communication System |
| OFDM | Orthogonal Frequency Division Multiplexing |
| PA | Power Amplifiers |
| PCCC | Parallel Concatenated Convolutional Code |
| PDR | Partial Dynamic Reconfiguration |
| PHY | Physical Layer |
| PL | Programmable Logic |
| PLB | Programmable Logic Blocks |
| PLCP | Physical Layer Convergence Protocol |
| PLL | Phase Locked Loop |
| PMD | Physical Medium Dependent |
| PPDU | PLCP Protocol Data Unit |
| PRC | Partial Reconfiguration Controller |
| PS | Processing System |
| PSDU | PHY Sub-Layer Service Data Units |
| QoS | Quality of Service |
| RF | Radio Frequency |
| RP | Reconfigurable Partition |
| SC-FDMA | Single-Carrier Frequency Division Multiple Access |
| S-DPCCH | Secondary Dedicated Physical Control Channel |
| SDR | Software Defined Radio |
| SF | Spreading Factor |
| TFCI | Transport-Format Combination Indicator |
| TPC | Transmit Power Control |
| TrCH | Transport Channel |

| | |
|---|---|
| TTI | Transmission Time Interval |
| UMTS | Universal Mobile Telecommunications System |
| USRP | Universal Software Radio Peripheral |
| WCDMA | Wide Code Division Multiple Access |
| WLAN | Wireless Local Area Network |

# Acknowledgments

This dissertation does not only hold the results of the year work, but also reflects the relationships with many generous and stimulating people. This is the time where we have the opportunity to present our appreciation to all of them.

First, to our advisors, Dr. Hassan Mostafa and Dr. Yasmine Fahmy, we would like to express our sincere gratefulness and appreciation for their excellent guidance, caring, patience, and immense help in planning and executing the work in a timely manner. Their great personality and creativity provided us with an excellent atmosphere for work, while their technical insight and experience helped us a lot in our research.

Of course, we will never find words enough to express the gratitude and appreciation that we owe to our families. Their tender love and support have always been the cementing force for building what we achieved. The all-round support rendered by them provided the much needed stimulant to sail through the phases of stress and strain.

We would like also to thank Eng. Sherif Hosny, who was always there for any research questions, providing precious advice and sharing his time and experience.

Finally, Thanks are due to previous GP members. Their theses were supportive, informative and guided us in many critical issues.

# Abstract

This graduation project discusses the implementation of chains for multi-standards communication (3G, LTE, and WIFI) on a dynamically and partially reconfigurable heterogeneous platform FPGA ZYNQ board. Implementation results highlight the benefit of considering an FPGA platform like (ZYNQ board) that supports efficiently intensive computation components.

The implementation of the desired chains for multi-standards communication proves the availability of Partial Dynamic Reconfiguration technology to support efficiently Software Defined Radio. This project aims to implement the transmitter and receiver chains for the three standards (Wi-Fi, 3G and LTE). Then reconfigure the FPGA by the desired chain on the fly without the need for resetting.

This technique depends on the new technology Partial Dynamic Reconfiguration (PDR) which is introduced by XILINX. The new technology is expected to save area, power and cost of communication devices and increases the speed of switching and reconfiguring the FPGA.

During the project, experience is gained in HDL & MATLAB modelling of the transmitter and receiver blocks of the three standards, building a system on chip that consists of: Micro-blaze processor IP, ICAP IP and system Ace IP to enable partial configuration and other peripherals. Thus enable the communication with PC while testing the reconfiguration on separate blocks and finally testing the reconfiguration of the entire standards chains.

CHAPTER 1

# Introduction

In this thesis we are going to prove the concept of the availability for using partial dynamic reconfiguration in implementing software defined radio. The thesis flow will go as following.

## 1.1 Organization of the thesis

Chapter 1 provides an overview of the communication system blocks, introduces the idea of the SDR followed by explaining briefly the PDR concept and discusses the project flow through the previous years.

Throughout Chapter 2, 3, 4, and 5 we will go deep in the implemented standards' transmitter and receiver (3G, Wi-Fi, LTE, and 2G). Those chapters introduce the architectures of the standards chains (Receiver and transmitter), implementation of the HDL codes, illustrating the challenges, mentioning each block implementation and any modification done. Each implementation is done giving the all data rates combinations, where complete system design is performed, obtaining different blocks' specifications and expected non-idealities.

Chapter 6 covers the testing part, functional verification of the chain blocks using MATLAB codes, sources of errors and fixed point simulation.

Chapter 7 covers the used FPGA (ZynQ board), the FPGA testing environment, I/O files and methods used to interface with PC (Hardware Implementation).

Chapter 8 discusses the PDR configuration methods, difference between them, their advantages and disadvantages, also, each chain implementation using one RP, switching between different chains.

Chapter 9 concludes our achievements and results through a year full of team work, enthusiasm, hard work and research. Also, it includes our conclusion and proposes the potentials expected in the upcoming years and improvements that are the next step for the projects fellows.

## 1.2 Motivation

In the last two decades, there has been tackling efforts from both the technological and industrial fields on how to increase the connectivity among people. The communication standards are being developed and upgraded to satisfy the speed and the time to handle connectivity among the users whose number is increasing with time. Consequently, this leads to the existence of different communication standards, but as a drawback the radio frequency spectrum is not utilized in an efficient way [1, 2], where the communication bands are not used simultaneously at the same time. The research in the radio spectrum utilization leads to two approaches to solve this problem, the first approach is the IA which is an antenna array technology that uses spatial beamforming and signal processing algorithms to cancel interference and reuse of the space resources [3]. IA depends on the DPC technique. The second approach is the CR which dynamically configures the user terminals, to utilize the radio spectrum that is not used, depending on the available wireless channels detected without interfering with the other users. In other words, CR is considered a way of managing the radio spectrum in an efficient way and it can be developed using the SDR technique [4, 5].

Generally, in a MSCS there exist two major problems, the utilization of the radio spectrum pointed to in the previous paragraph and utilization of hardware. As each standard has its own transceiver this leads to high cost, large area, high power consumption and low battery life. In the same time, the development of the central base stations and the users' devices changes tremendously to adapt to the new technologies and support the old ones. Developing hardware, upgrading and redistributing costs money and effort. These two major problems, unutilized radio spectrum and waste in hardware, lead to start searching to find a new way of reusing (reconfiguring) the same set of hardware to operate the old and new technologies. The utilization of the radio resources and the physical hardware resources can be done by offloading data transmitted between the different communication systems, and in the same time reconfiguring the hardware resources or reordering them to switch from a standard to another. The SDR is a way of radio system implementation using software, which is used to form different waveforms. These waveforms allow the system to switch among different communication standards. The motivation of the SDR came from the existence of some physical layer blocks

has the same functionality in the different communication systems like (GSM, UMTS, LTE, etc.…). Note that, these standards are not used at the same time which allows their hardware resources and radio spectrum resources to be used in a more efficient way. Also, the switching among the different waveforms should be dynamic, more or less in real time. The PDR is a technique used in the (FPGA), which allows hardware real time reconfigurable computing system. The PDR can be adopted using its capability of dynamically changing and partially configured, to implement real time SDR system.

## 1.3 Communication system

Figure 1-1 shows the main blocks in a modern communication system. It is composed of a DSP unit, digital and analog converters (DAC, ADC), RF front end and antenna. Because of achieving high data rates by processing communication signal digitally using software, which is more easily to develop, distribute and upgrade, the digital transceivers penetrates the traditional analog transceivers by pushing the digital and analog converters towards the antenna and pulling the communication systems more to software design on a given hardware. However, this work will concentrate on the DSP block whereas a brief description for each block is presented as follows:

- **Digital Signal Processing Block**: In the transmitter, this block is responsible for signal adaptation to be sent over a channel. Signal adaptation includes encryption, error correction coding schemes, modulation and further more. Whereas in the receiver this block is responsible for extracting the original information sent, by reconstructing the signal using demodulation, decoding and decryption. This block increases the flexibility of the radio development.
- **DAC/ADC Blocks**: Analog and digital converters used to transfer the signal between the analog domain and digital domain. Using ADC, the received signal is being digitized to be processed digitally using the DSP block. The digital representation depends on the sampling rate that leads to some information loss. While the DAC is reconstructing the signal to nearly the original one.
- **RF Front End Block**: It is the classical block that contains the LNA, filters and PA. Where this block is the most challenging block in the SDR development.

- **Antenna**: generally, the antenna is a passive device used to capture the electromagnetic waves from the surrounding media, and converts it to an electrical signal. The antenna design complexity varies from a single antenna to multiple antenna arrays. Where the smart antenna is an antenna array that uses the signal processing algorithms to locate the direction of signal arrival. And the reconfigurable antenna is capable of changing its frequency for adaptable systems.

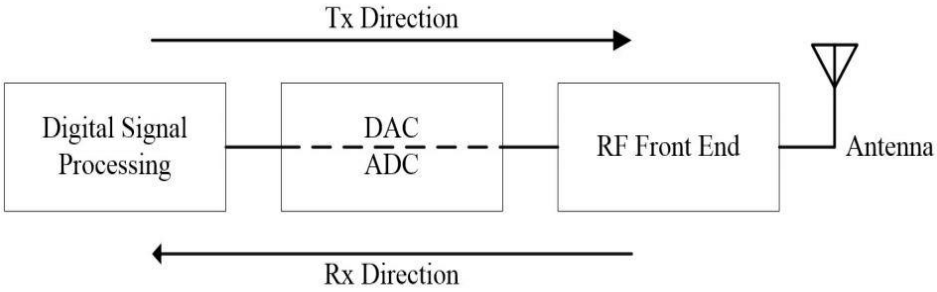


Figure 1-1: Simple communication system

## 1.4 Software Defined Radio (SDR)

### 1.4.1 SDR by Definition

The daily usage of communication standards is increasing. Phone calls, accessing the internet, sharing data and controlling devices are examples of modern communication usage. The devices held these standards vary in shape, functionality and the way of usage like mobile phones, wireless routers, smart chips, smart metering and even more. Although it is not easy to invent a generic device that can do everything, but it is achievable to manage the way of communication between them all. Zooming into this big picture to find adaptable communicating device to communicate the language (communication standard) of the other device. This adaptation is easier to be done through software defined modules, where these modules can change functionality by using the software. The SDR term defined by wireless innovation forum (formerly SDR forum) as “Radio in which some or all of the physical layer functions are Software Defined”.

### 1.4.2 Benefits and costs of SDR

There are many benefits of using SDR that it can be used in different industries and applications, hereby listing some of this advantages:

- **Interoperability**: An SDR can seamlessly communicate with multiple incompatible radios or act as a bridge between them. Interoperability was a primary reason for the US military's interest in, and funding of, SDR for the past 30 years. Different branches of the military and law enforcement use dozens of incompatible radios, hindering communication during joint operations. A single multi-channel and multi-standard SDR can act as a translator for all the different radios.

- **Efficient use of resources under varying conditions (Adaptability)**: An SDR can adapt the waveform to maximize a key metric. For example, a low-power waveform can be selected if the radio is running low on battery. A high-throughput waveform can be selected to quickly download a file. By choosing the appropriate waveform for every scenario, the radios can provide a better user experience.

- **Opportunistic frequency reuse (CR)**: An SDR can take advantage of underutilized spectrum. If the owner of the spectrum is not using it, an SDR can 'borrow' the spectrum until the owner comes back. This technique has the potential to dramatically increase the amount of available spectrum.

- **Updateability**: An SDR can be upgraded in the field to support the latest communications standards. This capability is especially important to radios with long life cycles such as those in military and aerospace applications. For example, a new cellular standard can be rolled out by remotely loading new software into an SDR base station, saving the cost of new hardware and the installation labor.

- **Costless**: An SDR can be adapted for use in multiple markets and for multiple applications. Economies of scale come into play to reduce the cost of each device. For example, the same radio can be sold to cell phone and automobile manufacturers. Just as significantly, the cost of maintenance and training is reduced.

- **Reusability**: The software modules implemented for a given standard for a product can be used with another product. That decreases the time and effort for building the same modules in other devices.

- **Remote upgrading**: Modules can be loaded and upgraded remotely without returning back to the lab.

The flexibility of the SDR comes with some disadvantages which are not related to the idea itself but it is related to the design complexity. It takes too much time and effort from engineers to develop different waveforms that can be adopted with the same set of hardware.

### 1.4.3 SDR platforms

The digital signal processing part in the communications system can be carried on different hardware platforms such as GPP, DSP and FPGA.

GPP is a microprocessor that is optimized for a powerful computing but consumes more power, it can be used in laboratories for research purpose.

DSP is a microprocessor that is less power consumption than GPP but its development is more difficult than the GPP, it is used in most of the cellular terminals and base stations.

FPGA is a microchip that can be configured by the user for a certain purpose, which makes it the best solution for implementing hardware blocks without unused logic gates.

## 1.5 Field Programmable Gate Arrays

One of the FPGA capabilities that is developed in the last decade is the PDR. This technique allows the FPGA to be configured partially and dynamically without switching off the system. This high flexibility of the FPGA allows it to be used in the hardware realization for SDR implementation. PDR helps in cost and resources reduction on the FPGA chip, providing a flexibility where a system can dynamically be configured without shutting it down. Moreover, this reconfiguration is done to a specific part instead of entire chip reconfiguration. Figure 1.2 illustrates a simple idea behind the PDR technique used in the modern FPGAs. Figure 1.2.a shows the

full configuration of the FPGA that an application consumes more area. Figure 1-2.b shows that the size of the application can be reduced by using PDR technique. i.e., if this application has different blocks not used at the same time so these modules can be time multiplexed. Each module can be loaded to function for a certain period of time then another module to be loaded. Figure 1-2.c shows that using PDR increases the size of the FPGA theoretically to realize more applications than regular FPGA configuration, this leads to more utilization of the FPGA resources. By applying the concept of the Partial Reconfiguration in the Software Defined Radio, it will result in a full reconfigurable wireless system which will be demonstrated through the thesis.

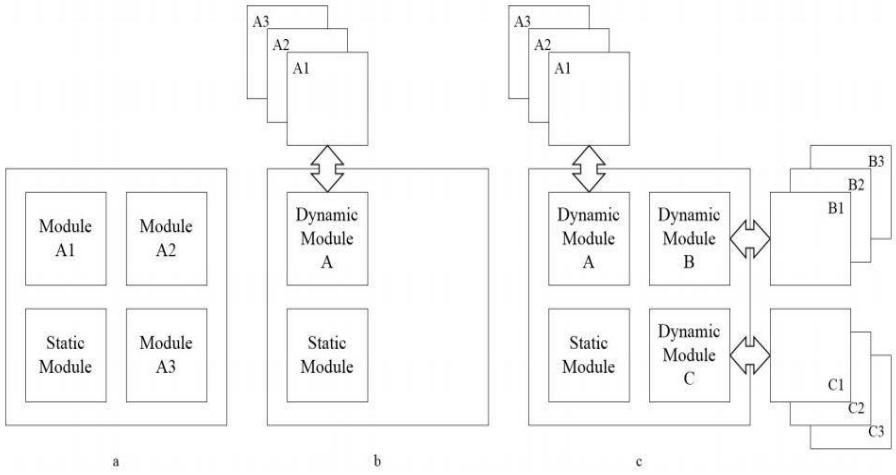


Figure 1-2: (a) Shows full FPGA configuration; (b) PDR technique to realize same system; (c) Shows how the FPGA size increased theoretically

### 1.5.1 FPGA VS ASIC

As described before, FPGA is a pre-manufactured silicon device with high flexibility and capability to be configured to realize different applications developed by a designer. They are programmed using HDL like VHDL or Verilog. So, it is naturally different from an ASIC, which is a circuit designed for a specific application with no reconfiguration capabilities.

An ASIC not only lacks the configurability feature, but also requires a long design cycle, and high start-up engineering cost compared to an FPGA. On the other side, FPGAs trade the extra area, power consumption, and delay for its unique feature. “Typically, FPGAs occupy larger area and dissipate more switching power than ASIC standard cells by factors of 20-30x and 10x, respectively.

From marketing prospective, FPGAs are used for small volume products need to be sold faster, where ASICs are used for large volume products, but the non-recurring engineering cost in the ASICs make their cost as a function of production volume in a flatter way than the FPGA. Figure 1-3 shows that the FPGA units used in market shifted to higher volumes domain. The reduction in the FPGA values is due to extending the FPGA functionality through reconfiguration.

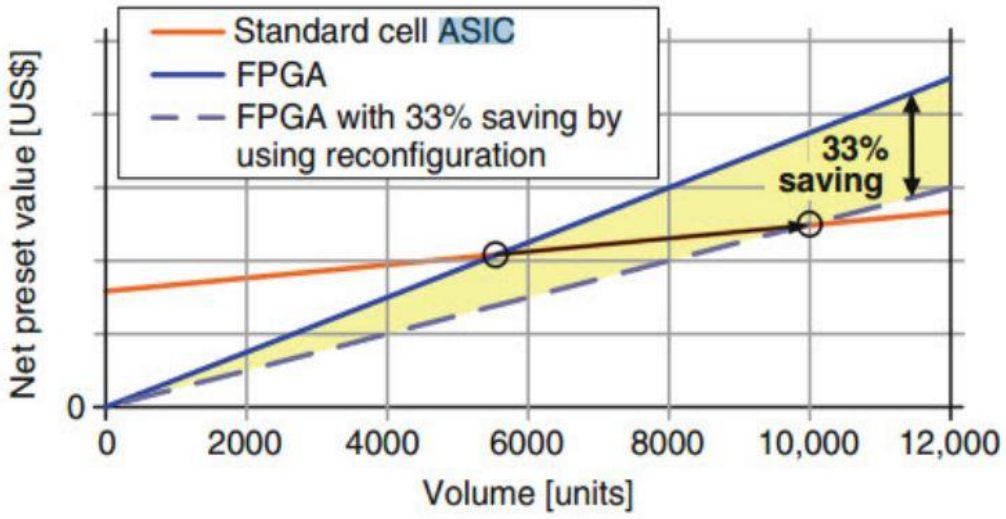


Figure 1-3: Value vs volume for ASIC and FPGA [6]

### 1.5.2 What is inside the FPGA?

**CLBs:** includes Registers and LUTs. They are the building blocks of the FPGA. Each Xilinx ZynQ device contains arrays of CLBs, each ZynQ CLB has two slices, and each slice has four LUTs and four Flip Flops. Combinatorial logic is implemented using LUTs, they can implement any 6-input combinatorial function of the user choice, with a cost of delay. Noting that, complexity of combinatorial function does not matter as long as it depends on six inputs or less. Flip Flops can be programmed to be latch, SR, JK, or D Flip Flops. Also, one carry chain is available per slice for arithmetic purpose; it helps to secure fast propagation of carry bit to nearby cells, which means it improves the speed. Moreover, it saves LUTs.

**Dedicated Blocks:** Like DSPs, which acts as an arithmetic logic unit, RAM blocks, PCIe core.

**Input/output Blocks**: with programmable standard functionality, like LVCMOS, LVPECL, and PCI. In fact, each bank can support several standards as long as they share the same reference voltage, or output voltage.

**Routing:** a combination of programmable and dedicated routing lines, use switching matrices connect lines from any source to any destination. Constrains can be applied.

**Clocking Resources:** like PLL which removes clock errors, and DCM. The dedicated clock trees balance the skew and minimize the delay. Thirty-two separate clock networks are available in ZynQ FPGA as shown in Figure 1-4.

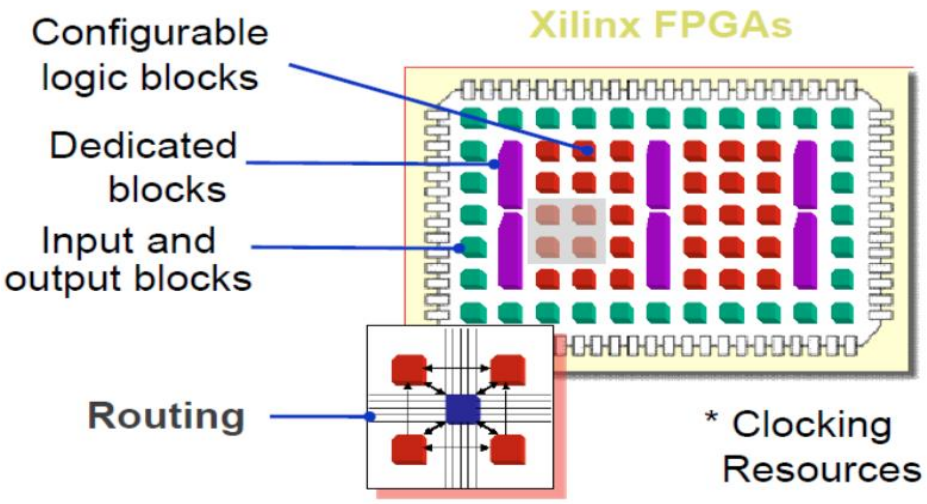


Figure 1-4: FPGA Internal structure

## 1.6 Project flow through the previous years

### 1.6.1 Internship Summer 2014

This was the first attempt in implementing SDR consisting. Most of the chain blocks of the three different chains (3G, Wi-Fi and LTE) were eliminated to make it simpler to implement what is important to prove the concept of PDR used to develop an SDR.

The chosen blocks were only implemented using VHDL & they were as following:

- In 3G: Convolutional Encoder, Rate Half & Rate Third.
- In Wi-Fi: Convolutional Encoder, Rate Half.

- In LTE: Convolutional Encoder, Rate Third.

### 1.6.2 Graduation Project 2015

In this year, the objective was to design, simulate, and implement DPR system for SDR on FPGAs which was met by investigating and modeling on two different steps.

The first step by implementing PDR system for convolutional encoders used in different communication standards 3G, LTE and WIFI (completing the internship work). Where the convolutional encoders initially not exist on the chip but stored in external memory and loaded on demand. This PDR design for the convolutional encoder was compared to conventional convolutional encoder system, where all encoders existed on the same chip. They were compared with respect to area, power, latency and memory. The results showed that PDR implementation consumes less power and area when compared to the normal design. Whereas the normal design had less memory and latency.

The second step was to implement ideal communication chains for 3G, LTE and WIFI using PDR technique where swapping occurs among different blocks for implemented encoders, modulation, FFT and DFT used in these standards. This produces a reconfigurable system that can adapt different communication standards. Using PDR shows an improvement in area and power consumption with fewer extra memory and latency when compared to the normal static implementation.

These designs of both steps were implemented on Xilinx FPGA kit XUPV5-LX110T.

### 1.6.3 Graduation Project 2016

In this year, the progress continued & the following results were achieved.

- HDL and MATLAB implementation of 3G full transmitter and some of receiver blocks.
- HDL and MATLAB implementation of WI-FI full transmitter.
- HDL and MATLAB implementation of some of LTE transmitter blocks.
- Building a test framework to Verify of HDL implementation.
- Implementation of the three chains on the FPGA (Virtex 5).

- Generating and proving the concept of multiple RPs by implementing it on a simple example.
- Debugging the FPGA results using Chipscope.
- Building a system on chip (SOC) with input and output files.
- Reducing the total area and resources needed for implementation of the three standards.
- Reducing the total power of the system as they eliminated the static and sleep mode power consumed by the idle chains.
- Reducing reconfiguration overhead by reconfigure each internal block of the chain after finishing its function. This is a kind of pipelining as there wasn't any need to wait until all frame data was generated to reconfigure each internal block of the chain.

CHAPTER 2

# 3G Chain

## 2.1 Introduction

By the late 1990s, the very success of GSM was again raising questions about the future need for yet more spectrum. The GSM community was initially focused on developing GSM circuit and packet switched data services. It was limited to maximum data rates of less than 50 kbps and neither can support video telephony. There was an obvious potential evolution towards a wider bandwidth CDMA system. The aim was to develop a radio system capable of supporting up to 2 Mbps data rates. The global WCDMA specification activities were combined into a 3GPP that aimed to create the first set of specifications by the end of 1999, called Release 99.

The early WCDMA networks offered some benefits for the end users including data rate up to 384 kbps in uplink and in downlink and simultaneous voice and data then the HSPA network has appeared where its efficiency has improved considerably especially with Ethernet-based transport and compact new base stations with simple installation, low power consumption and fast capacity expansion. HSPA evolution also includes a number of features that can enhance the spectral efficiency. QoS differentiation is utilized to control excessive network usage to keep users happy also during the busy hours. HSPA evolution includes features that cut down the power consumption considerably and also improve the efficiency of small packet transmission in the HSPA radio networks**.**

### 2.1.1 Physical layer & Frame structure

There are seven types of uplink dedicated physical channels, the uplink DPDCH, the uplink DPCCH, the uplink S-DPCCH, the uplink DPCCH2, the uplink E-DPDCH, the uplink E-DPCCH and the uplink HS-DPCCH [7].

Each frame consists of 5 sub-frames; each sub-frame consists of 3 slots so the frame consists of 15 slots. The Length of the frame corresponds to 38400 chips. Chip Rate is 3.84 Mcps so Frame Length is 10ms as shown in Figure 2-1 [7].

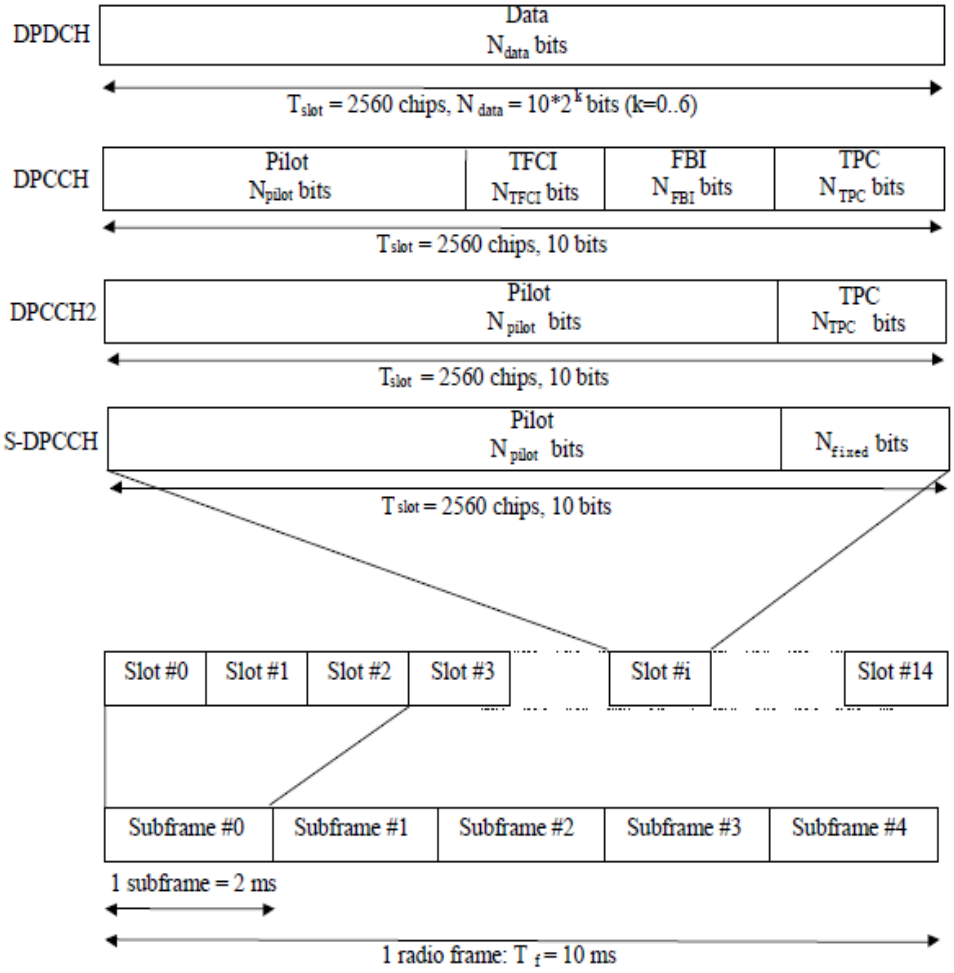


Figure 2-1: 3G Frame structure

The length of a Slot of DPDCH corresponds to 2560 chips. $N_{data}$ (number of data bits in the slot) =10 * $2^k$. The parameter k is related to the spreading factor SF where SF = 256 / $2^k$. In uplink: SF range is from 256 down to 4 so k range is from 6 down to 0 so $N_{data}$ range is from 10 to 640 bits. Slot of DPCCH: fixed SF of 256 and contains 10 bits. DPCCH has four fields: Pilot, TFCI, FBI, and TPC, size of each field is not fixed and defined in the table shown in Figure 2-2 [8].

| Slot Format #i | Channel Bit Rate (kbps) | Channel Symbol Rate (ksps) | SF | Bits/ Frame | Bits/ Slot | $N_{pilot}$ | $N_{TPC}$ | $N_{TFCI}$ | $N_{FBI}$ | Transmitted slots per radio frame |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 15 | 15 | 256 | 150 | 10 | 6 | 2 | 2 | 0 | 15 |
| 0A | 15 | 15 | 256 | 150 | 10 | 5 | 2 | 3 | 0 | 10-14 |
| 0B | 15 | 15 | 256 | 150 | 10 | 4 | 2 | 4 | 0 | 8-9 |
| 1 | 15 | 15 | 256 | 150 | 10 | 8 | 2 | 0 | 0 | 8-15 |
| 2 | 15 | 15 | 256 | 150 | 10 | 5 | 2 | 2 | 1 | 15 |
| 2A | 15 | 15 | 256 | 150 | 10 | 4 | 2 | 3 | 1 | 10-14 |
| 2B | 15 | 15 | 256 | 150 | 10 | 3 | 2 | 4 | 1 | 8-9 |
| 3 | 15 | 15 | 256 | 150 | 10 | 7 | 2 | 0 | 1 | 8-15 |
| 4 | 15 | 15 | 256 | 150 | 10 | 6 | 4 | 0 | 0 | 8-15 |
| 5 | 15 | 15 | 256 | 150 | 10 | 6 | 2 | 2* | 0 | 8-15 |

Figure 2-2: DPCCH Field

TFCI used for bit rate control, channel decoding, interleaving parameters for every DPDCH frame. FBI used for transmission diversity in the DL. TPC used for inner loop power control commands. TPC Bit Patterns are defined in Figure 2-3 [8].

| TPC Bit Pattern | | | Transmitter power control command |
|---|---|---|---|
| $N_{TPC} = 2$ | $N_{TPC} = 4$ | $N_{TPC} = 8$ | |
| 11 | 1111 | 11111111 | 1 |
| 00 | 0000 | 00000000 | 0 |

Figure 2-3: Bit patterns of TPC

## 2.2 3G Transmitter PHY Block Diagram

As discussed in section 1.6.3, the graduation project team was able to implement the HDL codes for all transmitter blocks but the HDL codes were not synthesizable as each block needed more resources than the available resources by the FPGA. The main resource consumer was the memories that exist within each block for either operational or synchronization purposes. This problem was solved by rewriting all memories HDL codes to match Xilinx Vivado HDL coding technique for memory without changing the memory interface within the block or the block functionality. Rewriting the codes allowed the Xilinx Vivado synthesizer to recognize the HDL code as RAM, either a block RAM or a dedicated RAM according to the memory size. In case of block RAM, a B-RAM resource is reserved. However, in case of dedicated RAM, a LUT resource is reserved and programed as RAM.

Transmitter of 3G consists of several blocks as shown in Figure 2-4. In the following sub-sections, each block of the chain is explained in more details _ illustrating its basic idea, showing its interfaces, connections, inputs & outputs & presenting its LUT utilization_.

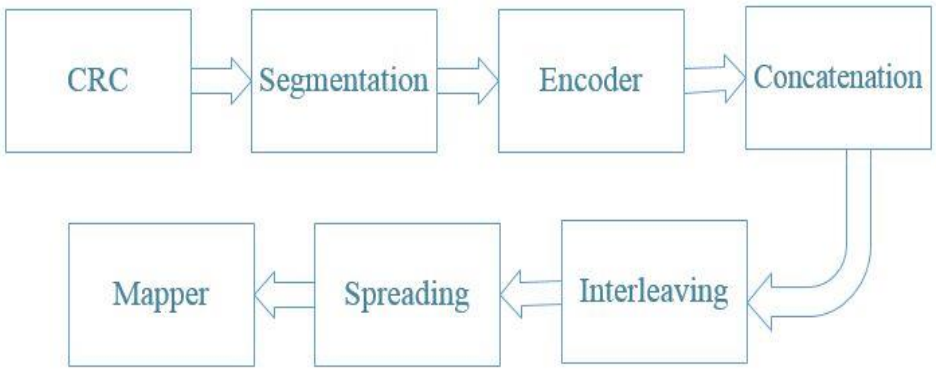


Figure 2-4: 3G Transmitter full chain blocks

### 2.2.1 CRC attachment

CRC process is provided on transport blocks for error detection in which the entire block is used to calculate the CRC parity bits for each transport block. Instead of adding just one bit to a block of data, several bits are added. The size of the CRC is 24, 16, 12, 8 or 0 bits and it is signaled from higher layers _depending on the channel_ what CRC size that should be used [9]. CRCs are typically implemented in hardware as a linear feedback shift register where its equations are shown in Table 2-1.

Table 2-1: Types of CRC

| CRC Mode | Equation |
|---|---|
| CRC24 | $g_{CRC24}(D) = D^{24} + D^{23} + D^6 + D^5 + D + 1$ |
| CRC16 | $g_{CRC16}(D) = D^{16} + D^{12} + D^5 + 1$ |
| CRC12 | $g_{CRC12}(D) = D^{12} + D^{11} + D^3 + D^2 + D + 1$ |
| CRC8 | $g_{CRC8}(D) = D^8 + D^7 + D^4 + D^3 + D + 1$ |

The top controlled module of CRC is as shown in Figure 2-5, A detailed CRC module is as shown in Figure 2-6, and the pins description of the controlled module is shown in Table 2-2.

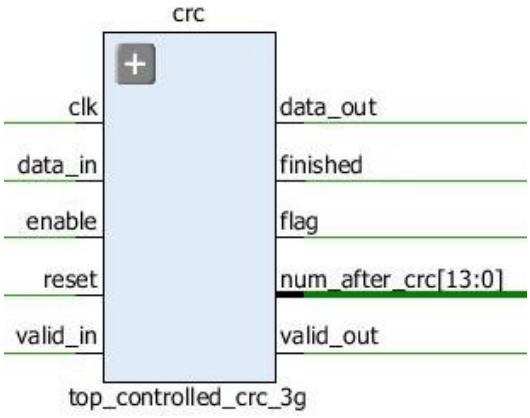


Figure 2-5: Top controlled CRC

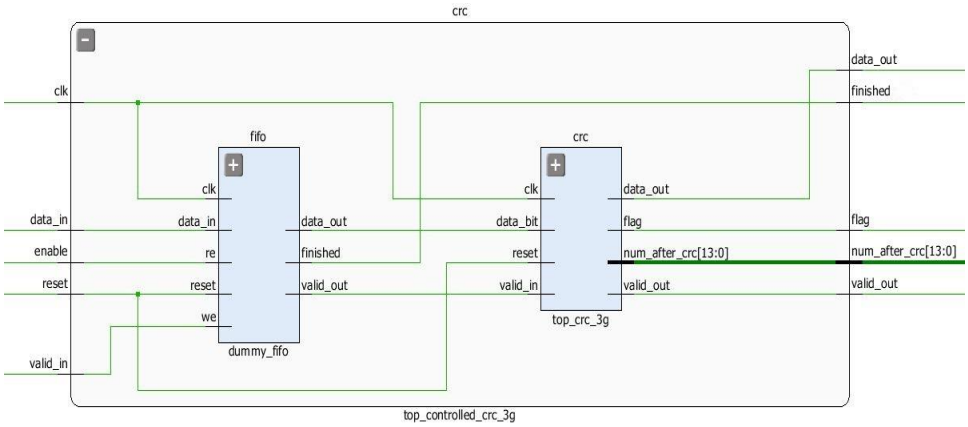


Figure 2-6: Top controlled CRC from inside

Table 2-2: pins description of CRC

| PIN | Description |
|---|---|
| data_in | The input bits to the block |
| data_out | The output data of the block |
| enable | This signal indicates that the next block (Segmentation) is ready to have data |
| finished | This signal indicates that the CRC block is ready for a new frame |
| Flag | This signal indicates that the num_after_crc is valid |
| num_after_crc | This signal indicates that total number of bits after CRC |
| valid_in | This signal indicates that the current data_in is valid data |
| valid_out | This signal indicates that the current data_out is valid data |

Finally, the LUT utilization of the CRC block is shown in Figure 2-7.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 139 | 0 | 53200 | 0.26 |
| LUT as Logic | 137 | 0 | 53200 | 0.26 |
| LUT as Memory | 2 | 0 | 17400 | 0.01 |
| LUT as Distributed RAM | 2 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 97 | 0 | 106400 | 0.09 |
| Register as Flip Flop | 97 | 0 | 106400 | 0.09 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 2-7: CRC LUT utilization

### 2.2.2 Segmentation

Segmentation of the bit sequence from transport block concatenation is performed if $X_i$ (the number of bits input to the segmentation) > Z (for convolutional coding: Z = 504 & for turbo coding: Z = 5114). The code blocks after segmentation are of the same size. The number of code blocks on TrCH 'i' is denoted by $C_i$. If, $X_i$, is not a multiple of Ci, filler bits are added to the beginning of the first block where the filler bits are always set to 0. Number of code blocks is gotten through the following relation: $C_i = \left\lceil \frac{X_i}{Z} \right\rceil$ [9].

The top controlled module of Segmentation is as shown in Figure 2-8, A detailed Segmentation module is as shown in Figure 2-9, and the pins description of the controlled module is shown in Table 2-3.

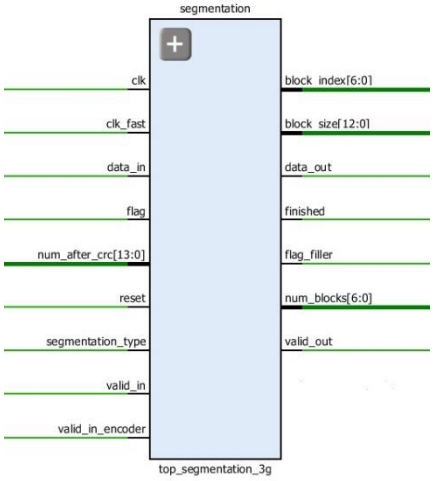


Figure 2-8: Top controlled Segmentation

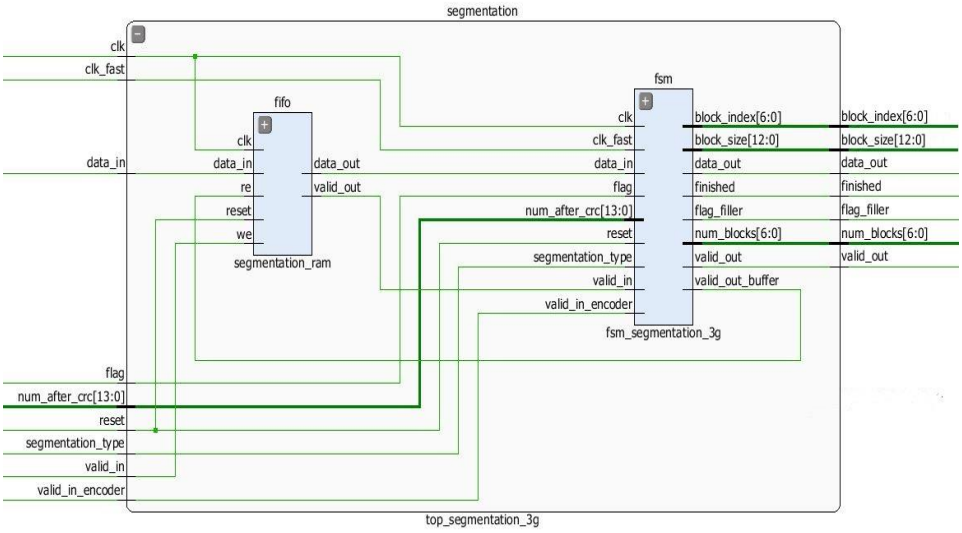


Figure 2-9: Top controlled Segmentation from inside

Table 2-3: pins description of Segmentation

| PIN | Description |
|---|---|
| block_index | This signal indicates the index for the block being transmitted to the encoder |
| block_size | This signal indicates the number of bits included in each block after performing the segmentation process |
| clk_fast | Connected to the clk_spreading signal to increase the speed for the division required to generate the number of blocks produced |
| data_in | The input bits to the block |
| data_out | The output bits of the block |
| finished | This signal indicates that the segmentation block is ready for the new frame |
| flag | This signal from the CRC indicates that the num_after_crc is ready to be read by the segmentation block |
| flag_filler | This signal is used for the encoder such that it does not read the extra zero filler bit that remains on the bus while moving from state to another inside the code. Consequently, this reserve that valid_out signal to remain always one within the data block |
| num_after_crc | This signal from the CRC indicates the total number of data bits plus concatenated CRC bits |
| num_blocks | This signal indicates the total number of blocks output from the segmentation process |
| segmentation_type | This signal is used to differentiate between Convolutional Encoder "0" or Turbo Encoder "1" |
| valid_in_encoder | This signal indicates that the next block (Encoder) is ready to have data |
| valid_in | This signal indicates that current data_in is valid data |
| valid_out | This signal indicates that current data_out is valid data |

Finally, the LUT utilization of the Segmentation block is shown in Figure 2-10.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 2840 | 0 | 53200 | 5.34 |
| LUT as Logic | 2840 | 0 | 53200 | 5.34 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 118 | 0 | 106400 | 0.11 |
| Register as Flip Flop | 99 | 0 | 106400 | 0.09 |
| Register as Latch | 19 | 0 | 106400 | 0.02 |
| F7 Muxes | 53 | 0 | 26600 | 0.20 |
| F8 Muxes | 4 | 0 | 13300 | 0.03 |

Figure 2-10: Segmentation LUT utilization

### 2.2.3 Encoder

Convolutional codes with constraint length 9 and coding rates 1/3 and 1/2 are defined. The configuration of the convolutional coder is presented in Figure 2.11, Figure 2.12. Output from the rate 1/3 convolutional coder shall be done in the order output0, output1, output2, output0, output1, output 2, output 0, ..., output 2. Output from the rate 1/2 convolutional coder shall be done in the order output 0, output 1, output 0, output 1, output 0... output 1 [9].

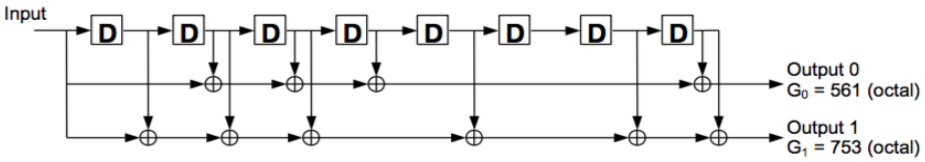


Figure 2-11: Rate 1/2 Convolutional encoder

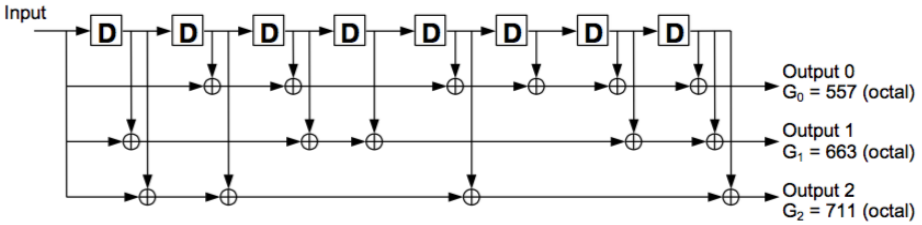


Figure 2-12: Rate 1/3 Convolutional encoder

For correct operation, 8 tail bits with binary value 0 shall be added to the end of the code block before encoding. Also, the initial value of the shift register of the coder shall be "all 0" when starting to encode the input bits.

The top controlled module of Encoder is as shown in Figure 2-13, A detailed Encoder module is as shown in Figure 2-14, and the pins description of the controlled module is shown in Table 2-4.

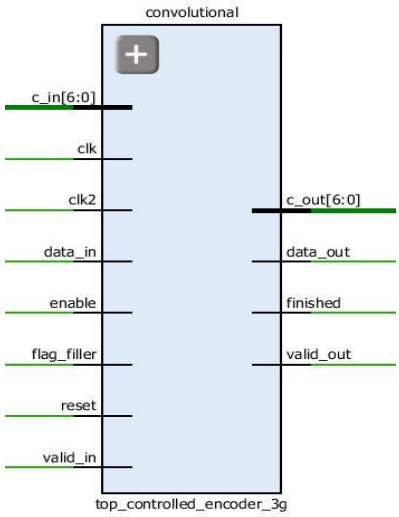


Figure 2-13: Top controlled Encoder

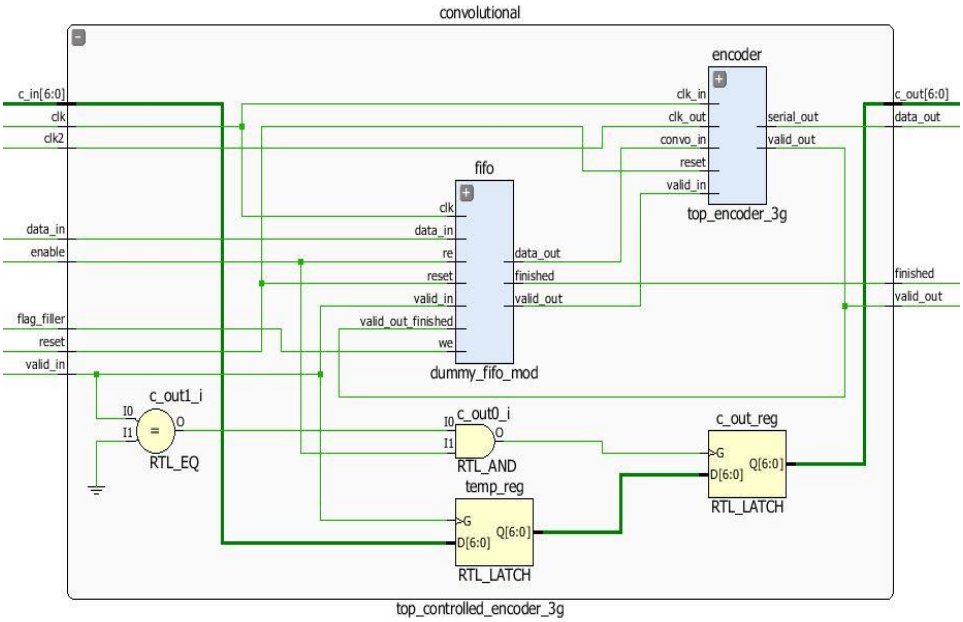


Figure 2-14: Top controlled Encoder from inside

Table 2-4: pins description of Encoder

| PIN | Description |
|---|---|
| c_in | This signal indicates the number of code blocks from the segmentation |
| c_out | This signal indicates the number of code blocks |
| clk2 | Clock of the serial output from the block |
| data_in | The input bits to the block |
| data_out | The output bits of the block |
| enable | This signal indicates that the next block (Concatenation) is ready to have data |
| finished | This signal indicates that the encoder block is ready for the new frame |
| flag_filler | This signal from segmentation |
| valid_in | This signal indicates that current data_in is valid data |
| valid_out | This signal indicates that current data_out is valid data |

Finally, the LUT utilization of the Encoder block is shown in Figure 2-15.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 84 | 0 | 53200 | 0.16 |
| LUT as Logic | 84 | 0 | 53200 | 0.16 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 75 | 0 | 106400 | 0.07 |
| Register as Flip Flop | 61 | 0 | 106400 | 0.06 |
| Register as Latch | 14 | 0 | 106400 | 0.01 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 2-15: Encoder LUT utilization

### 2.2.4 Code block concatenation

The input bit sequence for the code block concatenation block are the sequences $e_{rk}$, for r = 0…., C-1 and k = 0…., $Er$-1. The output bit sequence from the code block concatenation block is the sequence $f_k$ for k = 0…., G-1 where if we set "k = 0 & r = 0" then by looping with condition "r < C" with incrementing the r value by one "r = r +1" & set "j = 0" then by looping with condition "j < $E_r$", with incrementing the k & j values by one "k = k +1 & j = j +1" & performing the following "$f_k = e_{rj}$". The code block concatenation consists of sequentially concatenating the rate matching outputs for the different code blocks [9].

The top controlled module of concatenation is as shown in Figure 2-16, and the pins description of the controlled module is shown in Table 2-5.

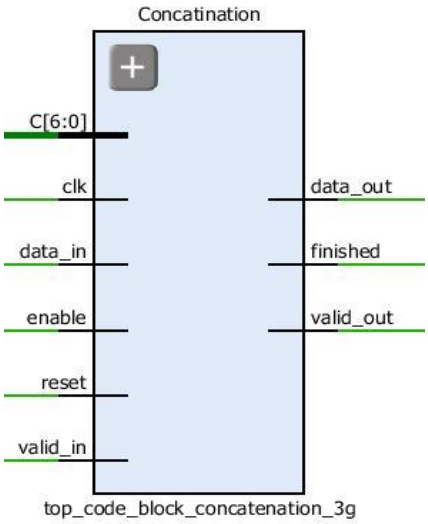


Figure 2-16: Top controlled Concatenation

Table 2-5: pins description of Concatenation

| PIN | Description |
|---|---|
| c | This signal indicates the number of code blocks from the segmentation block |
| data_in | The input bits to the block |
| data_out | The output bits of the block |
| enable | This signal indicates that the next block (Interleaver) is ready to have data |
| finished | This signal indicates that the Concatenation block is ready to have a new frame |
| valid_in | This signal indicates that current data_in is valid data |
| valid_out | This signal indicates that current data_out is valid data |

Finally, the LUT utilization of the Concatenation block is shown in Figure 2-17.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 97 | 0 | 53200 | 0.18 |
| LUT as Logic | 97 | 0 | 53200 | 0.18 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 72 | 0 | 106400 | 0.07 |
| Register as Flip Flop | 72 | 0 | 106400 | 0.07 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 2-17: Concatenation LUT utilization

### 2.2.5 Interleaving

Interleaving is a way to re-arrange data in a non-contiguous way to make it stand burst errors. These types of errors can destroy many bits in a row and make it hard to recover using FEC coding, since these expects the errors to be more uniformly distributed. This method is popular because it is a less complex and cheaper way to handle burst errors than directly increasing the power of the error correction scheme where interleaving causes increasing the performance of decoding [9].

The main disadvantage of using interleaving techniques is that increases latency because the entire interleaved block must be received before the packets can be decoded. Interleaving period equals to TTI which determines then number of columns in the interleaving matrix (10, 20, 40, 80ms => 1, 2, 4, 8 columns).

The top controlled module of Interleaving is as shown in Figure 2-18, A detailed Interleaving module is as shown in Figure 2-19, and the pins description of the controlled module is shown in Table 2-6.

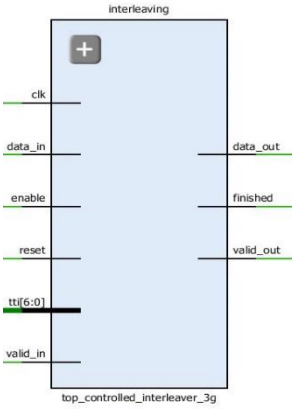


Figure 2-18: Top controlled Interleaving

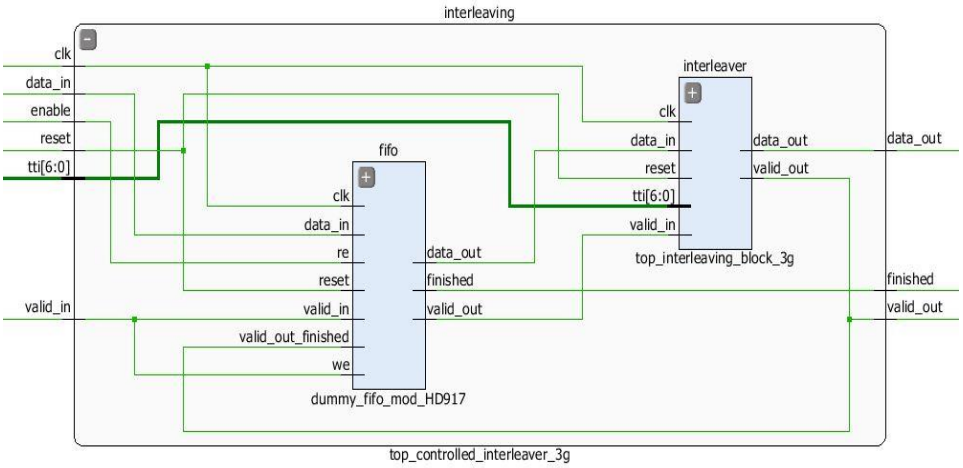


Figure 2-19: Top controlled Interleaving from inside

Table 2-6: pins description of Interleaving

| PIN | Description |
|---|---|
| data_in | The input bits to the block |
| data_out | The output bits of the block |
| enable | This signal indicates that the next block (Spreading & Scrambling) is ready to have data |
| finished | This signal indicates that the Interleaving block is ready to have a new frame |
| valid_in | This signal indicates that current data_in is valid data |
| valid_out | This signal indicates that current data_out is valid data |
| tti | Transmission Time Interval<br>possible values are: 10, 20, 40, and 80 |

Finally, the LUT utilization of the Interleaving block is shown in Figure 2-20.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 916 | 0 | 53200 | 1.72 |
| LUT as Logic | 916 | 0 | 53200 | 1.72 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 387 | 0 | 106400 | 0.36 |
| Register as Flip Flop | 387 | 0 | 106400 | 0.36 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 2-20: Interleaving LUT utilization

### 2.2.6 Spreading and Scrambling

Spreading is applied to the physical channels. It consists of two operations

- Channelization operation: increase the bandwidth of the signal using fully orthogonal codes called channelization codes to not interfere with each other. Every data is transformed into number of chips. The number of chips per data symbol is called the SF. The channelization codes are picked from the code tree as shown in Figure 2-21 [10].
- Scrambling operation: Scrambling code is applied to the spread signal and doesn't affect the signal bandwidth. The scrambling code can be a long code (a Gold code with 38400chips) or a short code (256 chips) the long code is used if the BS uses a Rake receiver and the short code is used if multiuser detector and interference cancellation receivers are used in BS.

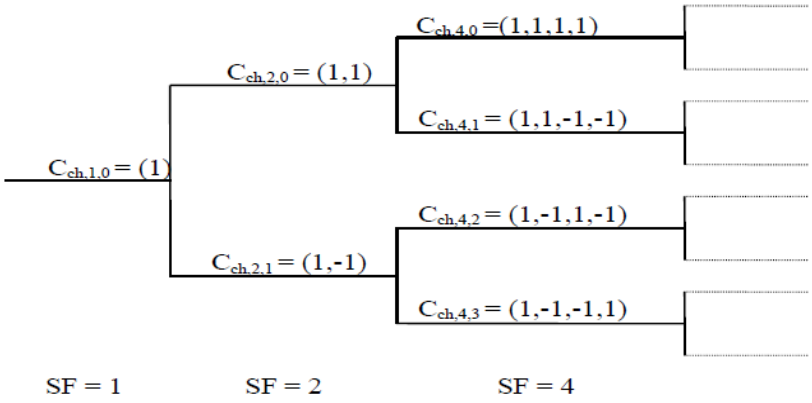


Figure 2-21: Code-tree for generation of Orthogonal Spreading Factor codes

The top controlled module of Spreading is as shown in Figure 2-22, A detailed Spreading module is as shown in Figure 2-23, and the pins description of the controlled module is shown in Table 2-7.

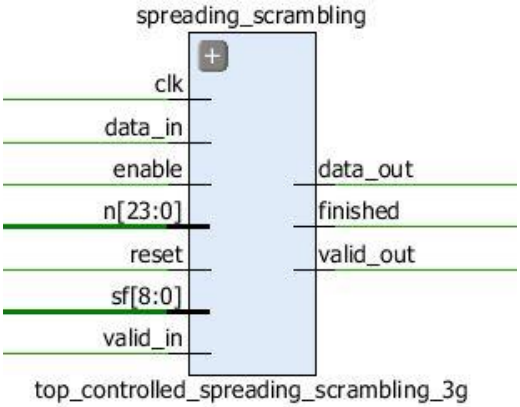


Figure 2-22: Top controlled Spreading

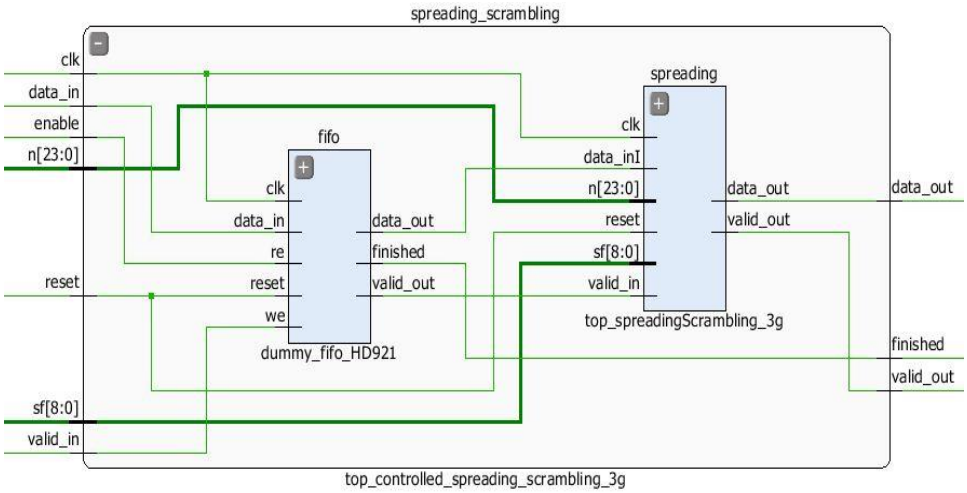


Figure 2-23: Top controlled Spreading from inside

Table 2-7: pins description of Spreading

| PIN | Description |
|---|---|
| data_in | The input bits to the block |
| data_out | The output bits of the block |
| enable | This signal indicates that the next block (Mapper) is ready to have data |
| finished | This signal indicates that the Spreading block is ready to have a new frame |
| n | The signal indicates scrambling sequence number |
| sf | The signal indicates the number of chips per data symbol |
| valid_in | The signal indicates the current data_in is valid data |
| valid_out | The signal indicates that current data_out is valid data |

Finally, the LUT utilization of the Spreading block is shown in Figure 2-24.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 148 | 0 | 53200 | 0.28 |
| LUT as Logic | 146 | 0 | 53200 | 0.27 |
| LUT as Memory | 2 | 0 | 17400 | 0.01 |
| LUT as Distributed RAM | 2 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 142 | 0 | 106400 | 0.13 |
| Register as Flip Flop | 118 | 0 | 106400 | 0.11 |
| Register as Latch | 24 | 0 | 106400 | 0.02 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 2-24: Spreading LUT utilization

### 2.2.7 Modulation (Mapper)

Modulation is the process by which information (e.g. bit stream) is transformed into sinusoidal waveform. A sinusoidal wave has three features those can be changed - phase, frequency and amplitude- according to the given information and to the used modulation technique. BPSK modulation technique is used according to the desired data rate. The bits are mapped to complex-valued modulation symbol d= (I + j Q). In BPSK, a single bit is mapped to a complex-valued modulation symbol according to Table 2-8 [10].

Table 2-8: BPSK mapping

| $b(i)$ | $I$ | $Q$ |
|---|---|---|
| 0 | $1/\sqrt{2}$ | $1/\sqrt{2}$ |
| 1 | $-1/\sqrt{2}$ | $-1/\sqrt{2}$ |

The top controlled module of Modulation is as shown in Figure 2-25, A detailed Modulation module is as shown in Figure 2-26, and the pins description of the controlled module is shown in Table 2-9.

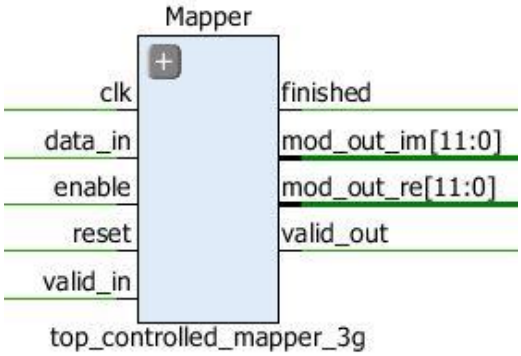


Figure 2-25: Top controlled Modulation

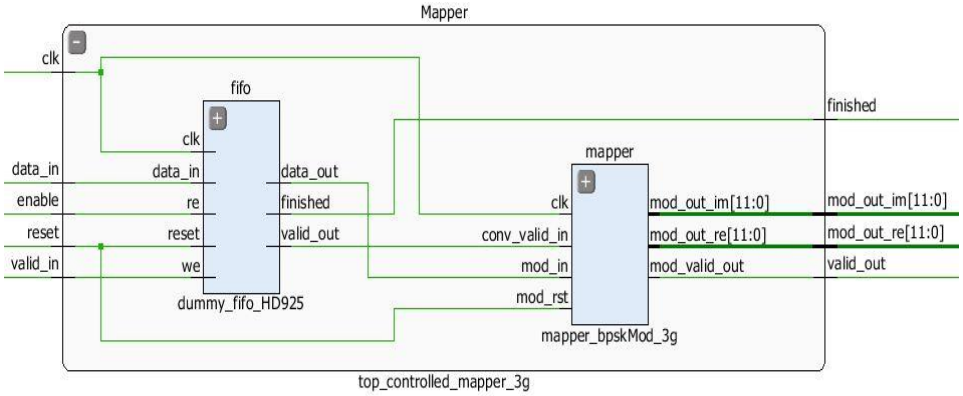


Figure 2-26: Top controlled Modulation from inside

Table 2-9: pins description of Modulation

| PIN | Description |
|---|---|
| data_in | The input bits to the block |
| enable | This signal indicates that the next block is ready to have data |
| finished | This signal indicates that the Mapper block is ready to have a new frame |
| mod_out_im | The modulated imaginary part of the input |
| mod_out_Re | The modulated real part of the input |
| valid_in | The signal indicates that current data_in is valid data |

Finally, the LUT utilization of the Modulation block is shown in Figure 2-27.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 71 | 0 | 53200 | 0.13 |
| LUT as Logic | 69 | 0 | 53200 | 0.13 |
| LUT as Memory | 2 | 0 | 17400 | 0.01 |
| LUT as Distributed RAM | 2 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 41 | 0 | 106400 | 0.04 |
| Register as Flip Flop | 41 | 0 | 106400 | 0.04 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 2-27: Modulation LUT utilization

The LUT utilization of the 3G transmitter full chain is shown in Figure 2-28.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 4568 | 0 | 53200 | 8.59 |
| LUT as Logic | 4562 | 0 | 53200 | 8.58 |
| LUT as Memory | 6 | 0 | 17400 | 0.03 |
| LUT as Distributed RAM | 6 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 940 | 0 | 106400 | 0.88 |
| Register as Flip Flop | 883 | 0 | 106400 | 0.83 |
| Register as Latch | 57 | 0 | 106400 | 0.05 |
| F7 Muxes | 63 | 0 | 26600 | 0.24 |
| F8 Muxes | 4 | 0 | 13300 | 0.03 |

Figure 2-28: 3G transmitter full chain LUT utilization

## 2.3 3G Receiver PHY Block Diagram

The main target of the receiver is to retrieve the same data send before transmitter so it consists of the blocks shown in Figure 2-29. The same procedure is performed in the receiver blocks as in the transmitter blocks where in the following sub-sections, each block of the chain is explained in more details _ illustrating its basic idea, showing its interfaces, connections, inputs & outputs & presenting its LUT utilization_.

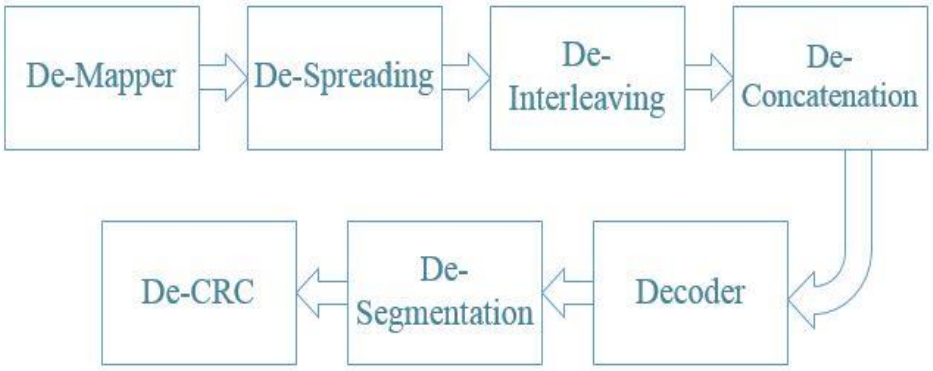


Figure 2-29: 3G Receiver full chain blocks

### 2.3.1 De-Modulation (De-Mapper)

It is the first block of the receiver that receives the real and imaginary data of the channel which came in the form of 12 bits divided to 9 bits representing the fraction part and 3 bits representing the real part. The main target of the block is to receive these data symbols, specify the decision region and convert these symbols to a stream of bits.

As discussed in section 2.2.7, In 3G we have only a BPSK mapper with a constellation not on the axis as shown in Table 2-8. So the equation of the decision region will be $y = -x$ where if $y > -x$ the output will be 0 and if $y < -x$ the output will be 1.

The top controlled module of De-Modulation is as shown in Figure 2-30, A detailed De-Modulation module is as shown in Figure 2-31, and the pins description of the controlled module is shown in Table 2-10.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 115 | 0 | 53200 | 0.22 |
| LUT as Logic | 99 | 0 | 53200 | 0.19 |
| LUT as Memory | 16 | 0 | 17400 | 0.09 |
| LUT as Distributed RAM | 16 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 60 | 0 | 106400 | 0.06 |
| Register as Flip Flop | 60 | 0 | 106400 | 0.06 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

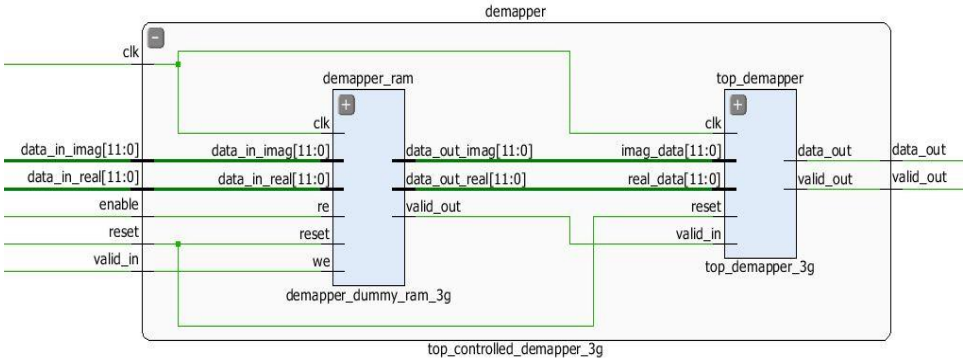


Figure 2-31: Top controlled De-Modulation from inside

Table 2-10: pins description of De-Modulation

| PIN | Description |
|---|---|
| data_in_imag | The imaginary part of the input bits to the block |
| data_in_real | The real part of the input bits to the block |
| data_out | The output data in the form of stream of bits |
| enable | This signal indicates that the next block (De-Spreading) is ready to have data |
| valid_in | The signal indicates that the current data_in either real or imaginary is valid data |
| valid_out | The signal indicates that current data_out is valid data |

Finally, the LUT utilization of the De-Modulation block is shown in Figure 2-32.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 115 | 0 | 53200 | 0.22 |
| LUT as Logic | 99 | 0 | 53200 | 0.19 |
| LUT as Memory | 16 | 0 | 17400 | 0.09 |
| LUT as Distributed RAM | 16 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 60 | 0 | 106400 | 0.06 |
| Register as Flip Flop | 60 | 0 | 106400 | 0.06 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 2-32: De-Modulation LUT utilization

### 2.3.2 De-Spreading and De-Scrambling

De-Spreading and De-Scrambling block consists of two operations

- Descrambling operation: one of the advantages of the scrambling codes that if we multiply the data with the scrambling data square we retrieve the same data. So in the descrambling process we multiply the data out from the De-Mapper with the same scrambling code of the transmitter by using the same scrambling sequence number (n).
- De-Spreading operation: multiply the data out from the De-Scrambler with a periodically repeated sequence of (1, 1, -1, 1) which is the same as the spreading code, we repeat these sequence with a number equal k where k= SF/4 this is because we transmit only one DPDCH. Then we integrate the data by increasing a signed register count when the output of multiplying with the spreading code is one and decreasing count when the output of multiplying with the spreading code is zero. So after we receive bits equal to SF we decide if the output will be 1 or 0 and we store this value in a data out register to be out while calculating the count and decide what the next bit is.

The top controlled module of De-Spreading is as shown in Figure 2-33, A detailed De-Spreading module is as shown in Figure 2-34, and the pins description of the controlled module is shown in Table 2-11.

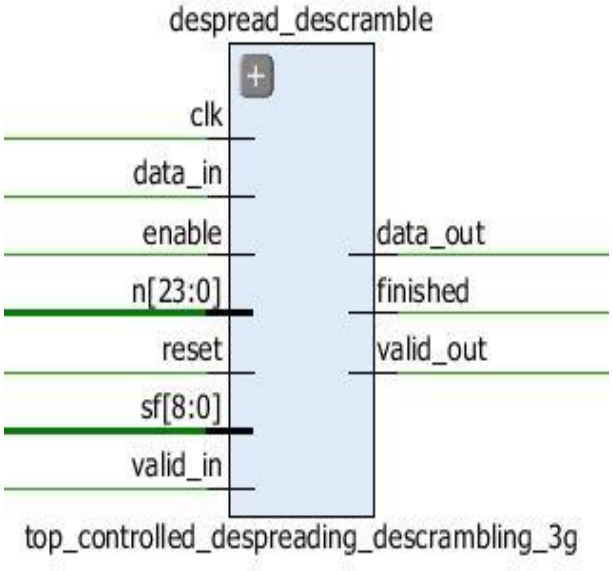


Figure 2-33: Top controlled De-Spreading

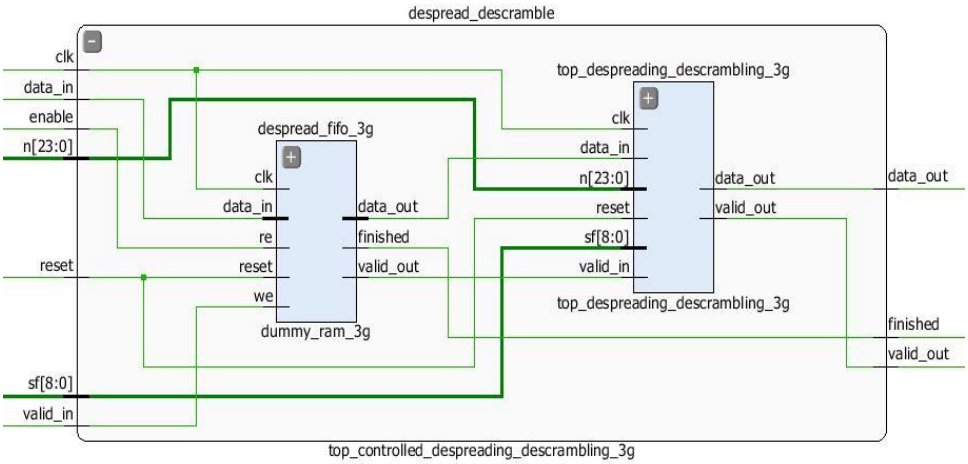


Figure 2-34: Top controlled De-Spreading from inside

Table 2-11: pins description of De-Spreading

| PIN | Description |
|---|---|
| data_in | The input bits to the block |
| data_out | The output data of the block |
| enable | This signal indicates that the next block (De-Interleaver) is ready to have data |
| finished | This signal indicates that the Spreading block is ready to have a new frame |
| n | This signal indicates the scrambling sequence number |
| sf | This signal indicates the number of chips per data symbol |
| valid_in | The signal indicates that the current data_in is valid data |
| valid_out | The signal indicates that the current data_out is valid data |

Finally, the LUT utilization of the De-Spreading block is shown in Figure 2-35.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 148 | 0 | 53200 | 0.28 |
| LUT as Logic | 146 | 0 | 53200 | 0.27 |
| LUT as Memory | 2 | 0 | 17400 | 0.01 |
| LUT as Distributed RAM | 2 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 142 | 0 | 106400 | 0.13 |
| Register as Flip Flop | 118 | 0 | 106400 | 0.11 |
| Register as Latch | 24 | 0 | 106400 | 0.02 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 2-35: De-Spreading LUT utilization

### 2.3.3 De-Interleaving

De-Interleaver is the block which re-arranges the received bits to repeal the impact of the Interleaver. The De-Interleaver block diagram is shown in Figure 2-36.

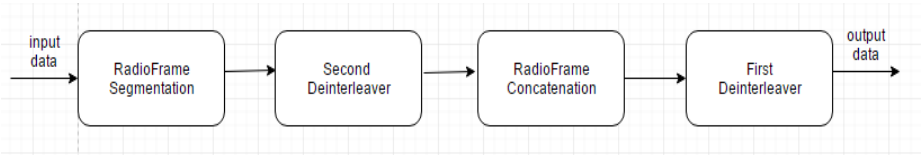


Figure 2-36: Block diagram of De-Interleaving

Radio Frame segmentation separates the input data into different frames depending on the value of "TTI". Table 12-2 shows the relationship between TTI and Number of frames.

Table 2-12: The relationship between TTI and Number of frames

| TTI | Number of frames |
|---|---|
| 10 | 1 |
| 20 | 2 |
| 40 | 4 |
| 80 | 8 |

Then every frame enters the second De-Interleaver to be re-arranged. It's to be noted that the second De-Interleaver is the reverse block of the second Interleaver in the transmitter where the number of columns of the interleaving matrix in the transmitter is 30. The columns of the matrix are numbered 0, 1, 2… 29 from left to

right. So, we add dummy bits in the receiver to make the data multiple of 30 to be the same as in the transmitter. Then, data with dummy bits is interleaved again using second Interleaver which is also implemented inside the main de-Interleaver. Finally, columns of the memory saving the interleaved data with dummy bits are arranged in the following arrangement

{col(0),col(12),col(25),col(6),col(18),col(3),col(15),col(26),col(9),col(22),
col(2),col(13),col(24),col(7),col(19),col(4),col(16),col(29),col(10),col(21),
col(1),col(14),col(27),col(8),col(20),col(5),col(17),ocl(28),col(11),col(23)}

Dummy bits are thrown out, and only data bits are getting out to the Radio frame concatenation.

Radio frame concatenation adds up the bits from different frames according to TTI to be fed to first de-Interleaver, as first de-Interleaver is inter-frame de-Interleaver.

In first de-Interleaver, number of columns is equal to number of frames. Data is written row by row, and then column permutation is done according to Table 2-13. Finally, data is read column by column.

Table 2-13: Columns arrangement in first de-Interleaving

| TTI | Permutation |
|---|---|
| 10 | Col (1) |
| 20 | Col (1), Col (2) |
| 40 | Col (1), Col (3), Col (2), Col (4) |
| 80 | Col (1), Col (5), Col (3), Col (7), Col (2), Col (6), Col (4), Col (8) |

The top controlled module of De-Interleaver is as shown in Figure 2-37, A detailed De-Interleaver module is as shown in Figure 2-38, and the pins description of the controlled module is shown in Table 2-14.

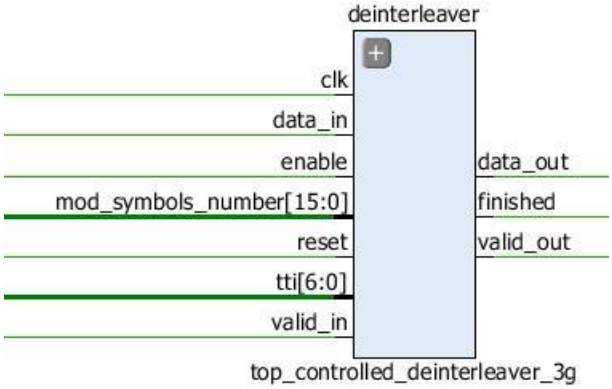


Figure 2-37: Top controlled De-Interleaving

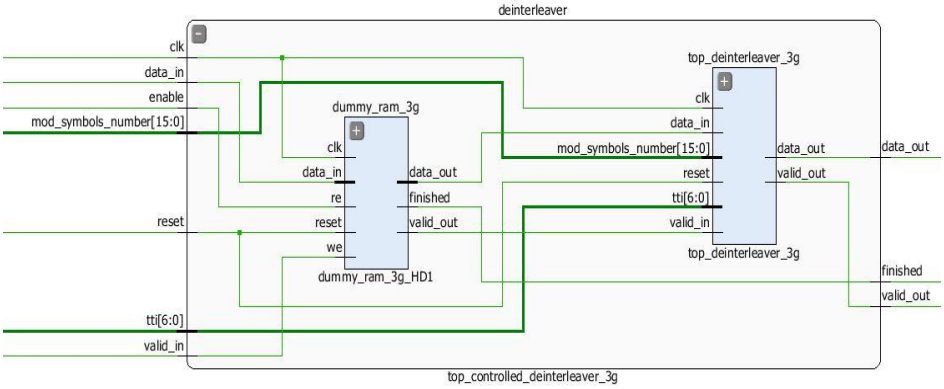


Figure 2-38: Top controlled De-Interleaving from inside

Table 2-14: pins description of De-Interleaving

| Pin | Description |
|---|---|
| data_in | The input data to the block |
| data_out | The output data of the block |
| enable | This signal indicates that the next block (De-Concatenation) is ready to have data |
| finished | This signal indicates that the De- Interleaver is ready to have data |
| valid_in | This signal indicates that current data_in is valid |
| valid_out | This signal indicates that current data_out is valid |
| tti | Transmission Time Interval |

Finally, the LUT utilization of the De-Interleaver block is shown in Figure 2-39.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 926 | 0 | 53200 | 1.74 |
| LUT as Logic | 924 | 0 | 53200 | 1.74 |
| LUT as Memory | 2 | 0 | 17400 | 0.01 |
| LUT as Distributed RAM | 2 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 444 | 0 | 106400 | 0.42 |
| Register as Flip Flop | 412 | 0 | 106400 | 0.39 |
| Register as Latch | 32 | 0 | 106400 | 0.03 |
| F7 Muxes | 5 | 0 | 26600 | 0.02 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 2-39: De-Interleaving LUT utilization

### 2.3.4 De-Concatenation

De-concatenation has the same concept as the Segmentation but with different z values where de-concatenation of the bit sequence is performed if $X_i$>Z. The code blocks after de- concatenation are of the same size.

To retrieve the same data before transmitter and as the data from segmentation is multiplied by the encoder rate so we calculate Z according to Table 2-15.

Table 2-15: how to calculate Z

| Z | Description |
|---|---|
| 504*2=1008 | Convolutional coding and coding rate = ½ |
| 5114*2=10228 | Turbo coding and coding rate = ½ |
| 504*3=1582 | Convolutional coding and coding rate = 1/3 |
| 5114*3=15342 | Turbo coding and coding rate = 1/3 |

The bits output from code block segmentation, for Ci ≠ 0, are denoted by $oir_1$, $oir_2$, $oir_3$… $oir_{ki}$ where i is the TrCH number, r is the code block number, and $K_i$ is the number of bits per code block.

Number of code blocks: $C_i = \left\lceil \frac{X_i}{Z} \right\rceil$ ,Number of bits in each code block (applicable for $C_i \neq 0$ only):

if Xi < 40 and Turbo coding is used, then $K_i = 40$

else $K_i = \left\lceil \frac{X_i}{C_i} \right\rceil$

end if

Number of filler bits: $Y_i = C_i * K_i - X_i$

--Insertion of filler bits

for k = 1 to $Y_i$ $Oi1_k = 0$ end for

for k = Yi+1 to Ki $Oi_k = Xi, (K-Y_i)$ end for

-- Segmentation (De-Concatenation)

r = 2

while $r \leq C_i$

for k = 1 to $K_i$ $Oir_k = X_i, (k+(r-1)-K_i-Y_i)$ end for

r = r+1

end while

The top controlled module of De-Concatenation is as shown in Figure 2-40, A detailed De-Concatenation module is as shown in Figure 2-41, and the pins description of the controlled module is shown in Table 2-16.

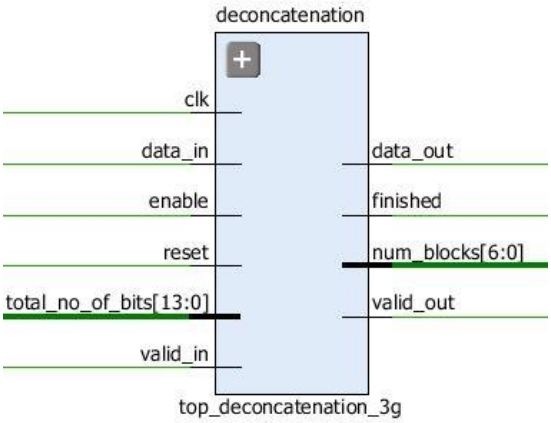


Figure 2-40: Top controlled De-Concatenation

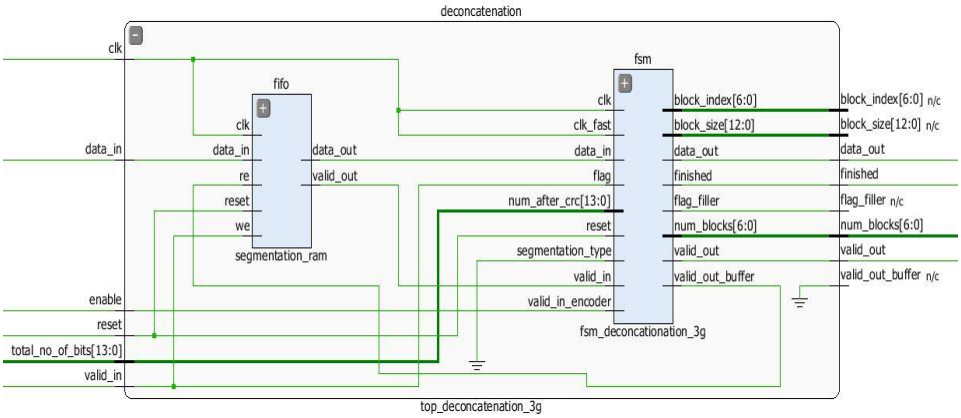


Figure 2-41: Top controlled De-Concatenation from inside

Table 2-16: pins description of De-Concatenation

| Pin | Description |
|---|---|
| data_in | The input data to the block |
| data_out | The output data of the block |
| enable | This signal indicates that the next block (Decoder) is ready to have data |
| finished | This signal indicates that the De-Concatenation is ready to have data |
| num_Blocks | This signal indicates the total number of blocks output from the de-concatenation process |
| total_no_of_bits | This signal indicates the total number of bits entering the block |
| valid_in | This signal indicates that current data_in is valid |
| valid_out | This signal indicates that current data_out is valid |

Finally, the LUT utilization of the De-Concatenation block is shown in Figure 2-42.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 2787 | 0 | 53200 | 5.24 |
| LUT as Logic | 2787 | 0 | 53200 | 5.24 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 118 | 0 | 106400 | 0.11 |
| Register as Flip Flop | 99 | 0 | 106400 | 0.09 |
| Register as Latch | 19 | 0 | 106400 | 0.02 |
| F7 Muxes | 53 | 0 | 26600 | 0.20 |
| F8 Muxes | 4 | 0 | 13300 | 0.03 |

Figure 2-42: De-Concatenation LUT utilization

### 2.3.5 Decoder

Decoder block is responsible for detecting the error in the received bit stream with the help of the redundant bits added by the encoder. After that decoder either requests retransmission or starts correcting received bit stream according to its type. Decoder starts correcting the received bit stream after detecting errors. Several algorithms exist to decode convolutional codes: trellis (Viterbi) decoders, sequential (Fano) decoders, stack decoders…etc. However, Viterbi decoders has optimum performance in terms of error correction, as it provides maximum likelihood (ML). Thus, in the decoder implemented for the 3G receiver chain, a Viterbi decoder is used.

Viterbi Algorithm (Maximum Likelihood Decoding)

- It is one type of Convolution Decoding. It is used to correct the error.
- This Viterbi Algorithm fully based on the Trellis structure.
- This Decoding has two major pads:

  -Metric Value: It's like a hamming distance i.e., number of bits to be changed and this value is added to a previous node metric value also this value is called metric value.

  - Survivor path or active path: In each node, after the 2nd stage, two paths will be entering. This node has two metric values. The path with lower

metric value is retained and the other path is discard. This retained path is called a survivor path or an active path.

Algorithm Steps

• Step (1): First stage and second stage is like a trellis. Finding the metric value for each node (Metric value to be found with help of received bits by comparing them with the coded bits). In Initial node metric value becomes zero. This step is illustrated in Figure 2-43.

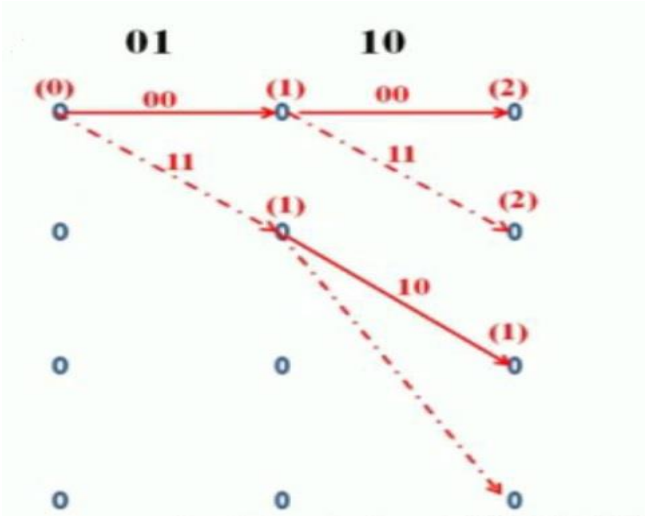


Figure 2-43: Algorithm Step 1

• Step (2): After the 2nd stage each node receiving two paths. Each path has a single metric value therefore after the second state each node has 2 metric values. This step is illustrated in Figure 2-44.

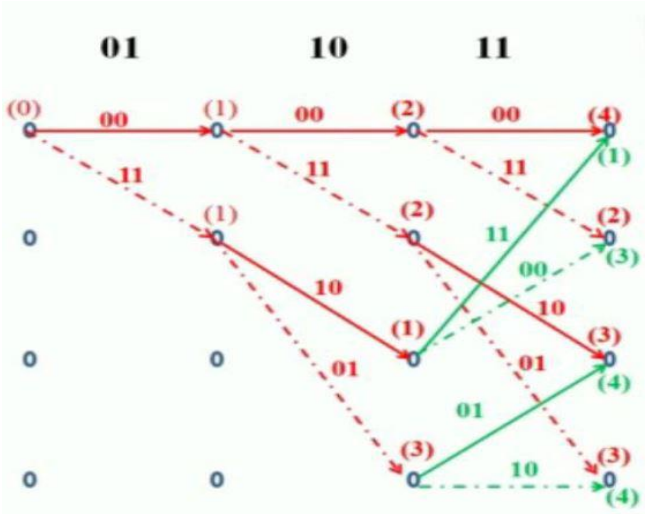


Figure 2-44: Algorithm Step 2

• Step (3): Finding the survivor path with help of minimum metric value (Minimum metric path only retained others are discard). This step is illustrated in Figure 2-45.

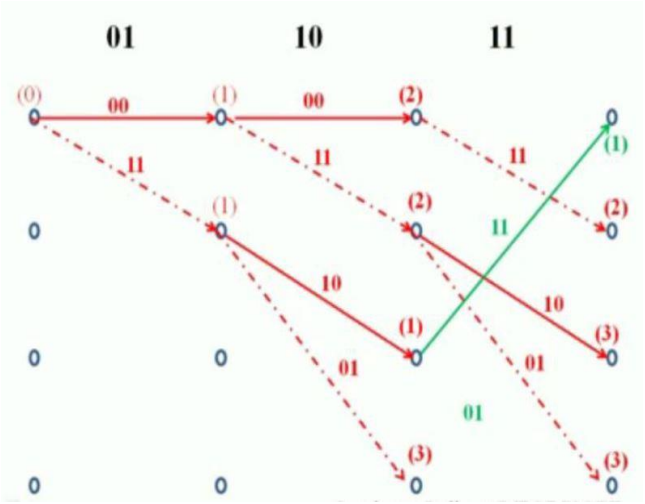


Figure 2-45: Algorithm Step 3

• Step (4): Step (2) and Step (3) are repeated until the received bit sequence stop. This step is illustrated in Figure 2-46.

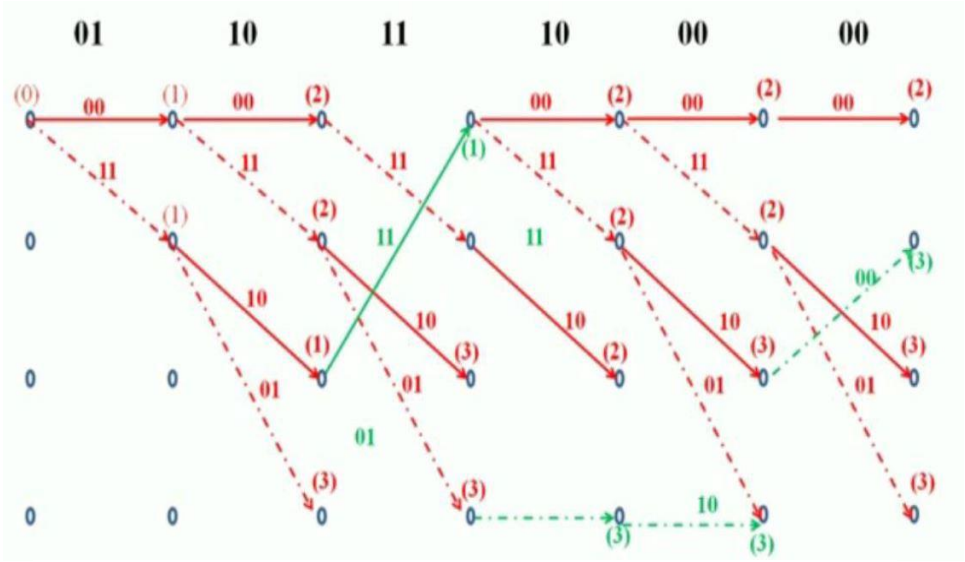


Figure 2-46: Algorithm Step 4

• Step (5): Finally find the active path. This active path code word is the corrected code word and find the original message sequence. This step is illustrated in Figure 2-47.

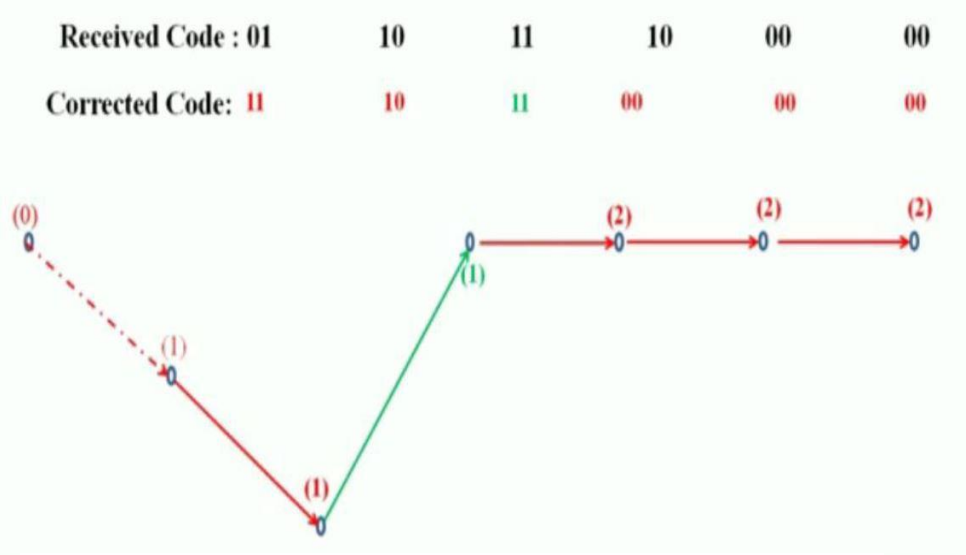


Figure 2-47: Algorithm Step 5

Hardware Modeling of Viterbi Algorithm:

In order to implement the pre-mentioned algorithm, the operation of the algorithm needs to be modelled to several hardware sub-blocks, each performs a certain function that contributes in the whole algorithm operation. The Viterbi algorithm is modelled as:

1- Serial to Parallel (S2P): This is implemented as a memory, which stores he input bit stream (serial bits) and outputs them in pairs of 2 bits (input code).

2- Branch Metric Generator (BMG): At this block the received data symbols are compared with the ideal outputs of the encoder from the transmitter to compute the hamming distance. Hamming distance is defined as the number of bits in the received data the does not match the ideal outputs of the encoder from the transmitter.

3- Add Compare Selection (ACS): This block adds the computed hamming distance, by BMG, to the pre-calculated path metric. Then, it compares between the path metrics of the paths entering the same node and selects the lowest.

4- Metric Memory: This memory is the memory that the path metrics are stored after each stage. It is composed of 2 RAMs one for storing and the

other for reading. At the end of the received bit stream, metric memory computes the state with the lowest path metric.

5- Survivor Memory: This memory is the memory that the survivor paths are stored after each stage. After the determination of the last state of the active path. It computes the previous states of the active path.

6- Trace Back: This block receives the last state in the active path. Then, it sends the state to the survivor memory and receives the previous state of the active path. After that, it computes the last bit in the corrected stream and saves it in a stack. This operation is repeated until the corrected bit stream is all stored in the stack. Then it starts reading for the stack to output the corrected bit stream.

7- Control Unit: All previous blocks' functions are controlled in timing and operation by this unit.

8- Clock conversion block: In some system, like 3G, encoder steps down the frequency of the output. Hence, decoder is expected to step up the frequency to preserve normal operation. This block is modelled as a memory at which data is store with normal clock of operation. However, the data will be read using the required clock of operation of the next block. This block also performs the operation of tail bits' removal for the implemented decoder.

This model is illustrated in Figure 2-48.

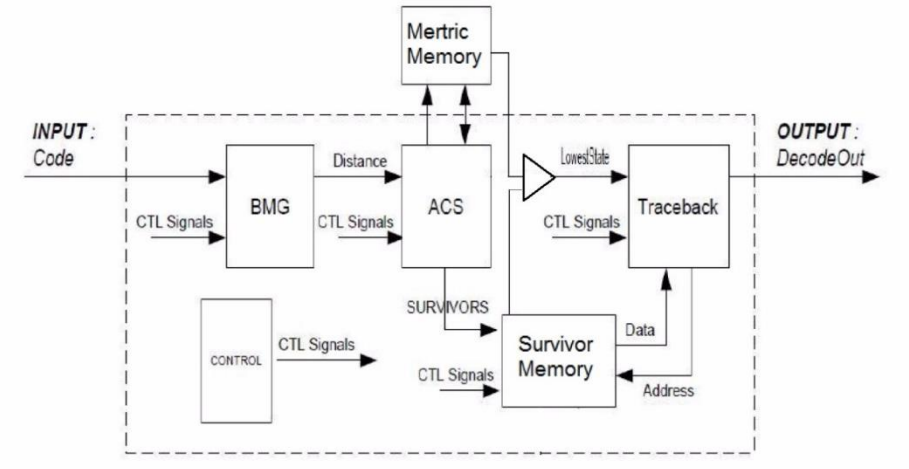


Figure 2-48: Hardware Modeling of Viterbi Algorithm

Sequence of HW operation:

1- Store data in S2P.

2- Get one input code word.

3- Calculating the hamming distance (branch metric).

4- Reading the previous path metric for all the states from the Metric Memory.

5- Add the branch metric to the path metric for the old state.

6- Compare the sums for paths arriving at the new state (there are only two such paths incoming).

7- Select the path with the smallest value which is called the survivor path. If both path Metrics are equal, then any one is chosen.

8- Writing the survivor path in survivor memory unit to be used in the trace back process.

9- Writing the new path metric in metric memory unit

10- When the sliding window reaches its end then begins the trace back process.

Design Specification:

In practical implementation of the Viterbi algorithm, it requires very big memories. So in order to decrease the size of memories required, the operation is segmented. At which, after few stages, a trace back operation is performed to compute the current active path. Hence, it clears the memory of the stored data and allows the algorithm to resume. The number of stages after which trace back is performed is referred to as trace-back length. Also, in order to decrease the latency of the decoder, states are modelled into 16 group. Each group contains 16 state.

- $clk_{min}$ for operation = 15 ns
- Code rate = ½

- Constraint length (k) = 9
- Number of states = 256
- Trace-back length = 64
- Survivor Memory size = 2k x Trace-back length = 512 x 64
- Metric Memory size = 2 x 192 x 16.
- Maximum input stream = 1024 bit.

The top controlled module of Decoder is as shown in Figure 2-49, and the pins description of the controlled module is shown in Table 2-17.

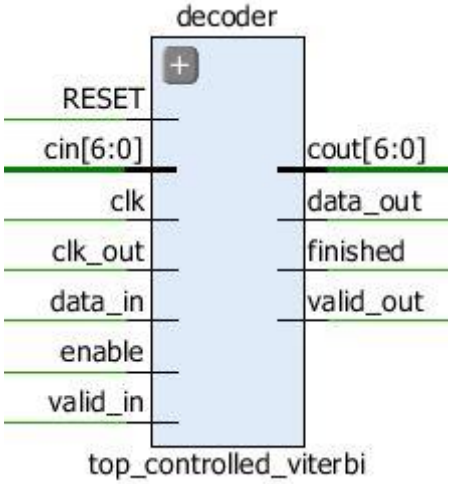


Figure 2-49: Top controlled Decoder

Table 2-17: pins description of Decoder

| Pin | Description |
|---|---|
| cin | It is the number of code blocks computed by de-concatenation |
| clk_out | It is the clock with which output bits should be sent to the next block |
| cout | It is the number of code blocks computed by de-concatenation. It is a 7-bit bus |
| data_in | The input data to the block |
| data_out | The output data of the block |
| enable | This signal indicates that the next block (De-Segmentation) is ready to have data |
| finished | This signal indicates that the De-Concatenation is ready to have data |
| valid_in | This signal indicates that current data_in is valid |
| valid_out | This signal indicates that current data_out is valid |

Finally, the LUT utilization of the Decoder block is shown in Figure 2-50.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 1790 | 0 | 53200 | 3.36 |
| LUT as Logic | 1261 | 0 | 53200 | 2.37 |
| LUT as Memory | 529 | 0 | 17400 | 3.04 |
| LUT as Distributed RAM | 529 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 582 | 0 | 106400 | 0.55 |
| Register as Flip Flop | 568 | 0 | 106400 | 0.53 |
| Register as Latch | 14 | 0 | 106400 | 0.01 |
| F7 Muxes | 43 | 0 | 26600 | 0.16 |
| F8 Muxes | 16 | 0 | 13300 | 0.12 |

Figure 2-50: Decoder LUT utilization

### 2.3.6 De-Segmentation

De-Segmentation has the same design & implementation as the concatenation block explained in section 2.2.4. The top controlled module of De-Segmentation is as shown in Figure 2-51, and the pins description of the controlled module is shown in Table 2-18.

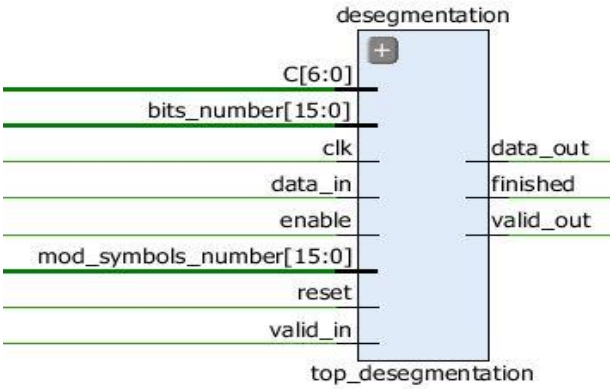


Figure 2-51: Top controlled De-Segmentation

Table 2-18: pins description of De-Segmentation

| Pin | Description |
|---|---|
| C | It is the number of code blocks computed by de-concatenation (decoder forwards it to the de-segmentation) |
| data_in | The input data to the block |
| data_out | The output data of the block |
| enable | This signal indicates that the next block (De-CRC) is ready to have data |
| finished | This signal indicates that the De-Segmentation is ready to have data |
| valid_in | This signal indicates that current data_in is valid |
| valid_out | This signal indicates that current data_out is valid |

Finally, the LUT utilization of the De-Segmentation block is shown in Figure 2-52.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 143 | 0 | 53200 | 0.27 |
| LUT as Logic | 143 | 0 | 53200 | 0.27 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 72 | 0 | 106400 | 0.07 |
| Register as Flip Flop | 72 | 0 | 106400 | 0.07 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 2-52: De-Segmentation LUT utilization

### 2.3.7 De-CRC

CRC check process (De-CRC) is provided for error check in which the entire received block is used to calculate the CRC parity bits for each received block.

We receive the total number of bits and subtract the CRC bits number from it and generate CRC parity bits by equations shown in Table 2-1 for only total number of bits _ CRC bits.

Finally, we compare these generated bits with the last bits received and decide out if this data was right or wrong. The data is wrong if there is any mismatch in the comparison.

The top controlled module of De-CRC is as shown in Figure 2-53, A detailed De-CRC module is as shown in Figure 2-54, and the pins description of the controlled module is shown in Table 2-19.

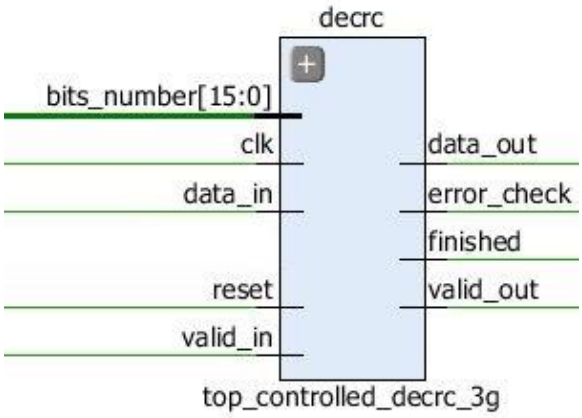


Figure 2-53: Top controlled De-CRC

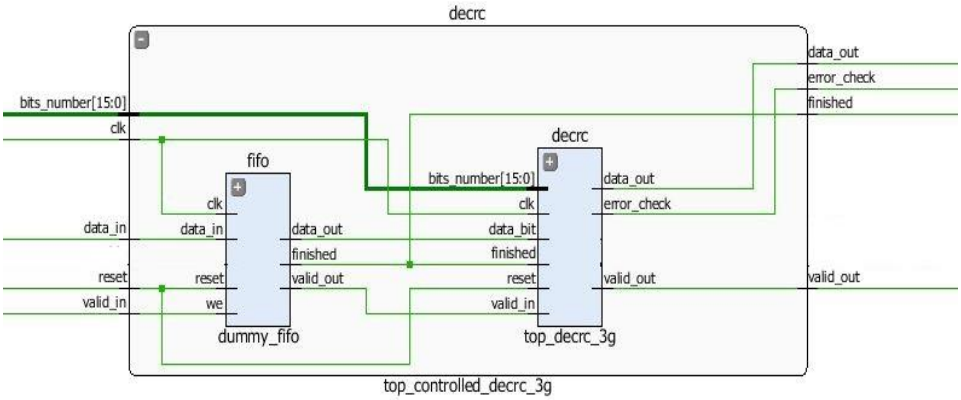


Figure 2-54: Top controlled De-CRC from inside

Table 2-19: pins description of De-CRC

| Pin | Description |
|---|---|
| bits_number | This signal indicates the total number of bits entering the block |
| data_in | The input data to the block |
| data_out | The output data of the block |
| error_check | This signal indicates the result of comparison between the generated bits from the block with the last bits received |
| finished | This signal indicates that the De-CRC is ready to have data |
| valid_in | This signal indicates that current data_in is valid |
| valid_out | This signal indicates that current data_out is valid |

Finally, the LUT utilization of the De-CRC block is shown in Figure 2-55.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 188 | 0 | 53200 | 0.35 |
| LUT as Logic | 186 | 0 | 53200 | 0.35 |
| LUT as Memory | 2 | 0 | 17400 | 0.01 |
| LUT as Distributed RAM | 2 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 116 | 0 | 106400 | 0.11 |
| Register as Flip Flop | 116 | 0 | 106400 | 0.11 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 2-55: De-CRC LUT utilization

The LUT utilization of the 3G receiver full chain is shown in Figure 2-56.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 6417 | 0 | 53200 | 12.06 |
| LUT as Logic | 5866 | 0 | 53200 | 11.03 |
| LUT as Memory | 551 | 0 | 17400 | 3.17 |
| LUT as Distributed RAM | 551 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 1579 | 0 | 106400 | 1.48 |
| Register as Flip Flop | 1490 | 0 | 106400 | 1.40 |
| Register as Latch | 89 | 0 | 106400 | 0.08 |
| F7 Muxes | 105 | 0 | 26600 | 0.39 |
| F8 Muxes | 20 | 0 | 13300 | 0.15 |

Figure 2-56: 3G receiver full chain LUT utilization

The utilization report of the 3G full chain is shown in Figure 2-57.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 10493 | 0 | 53200 | 19.72 |
| LUT as Logic | 9934 | 0 | 53200 | 18.67 |
| LUT as Memory | 559 | 0 | 17400 | 3.21 |
| LUT as Distributed RAM | 557 | 0 | | |
| LUT as Shift Register | 2 | 0 | | |
| Slice Registers | 2531 | 0 | 106400 | 2.38 |
| Register as Flip Flop | 2379 | 0 | 106400 | 2.24 |
| Register as Latch | 152 | 0 | 106400 | 0.14 |
| F7 Muxes | 128 | 0 | 26600 | 0.48 |
| F8 Muxes | 24 | 0 | 13300 | 0.18 |

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Block RAM Tile | 20 | 0 | 140 | 14.29 |
| RAMB36/FIFO* | 12 | 0 | 140 | 8.57 |
| RAMB36E1 only | 12 | | | |
| RAMB18 | 16 | 0 | 280 | 5.71 |
| RAMB18E1 only | 16 | | | |

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| DSPs | 5 | 0 | 220 | 2.27 |
| DSP48E1 only | 5 | | | |

Figure 2-57: 3G full chain utilization report

CHAPTER 3

# WI-FI Chain

## 3.1 Introduction

WLAN technology and the WLAN industry date back to the mid-1980s when the FCC first made the RF spectrum available to industry. During the 1980s and early 1990s, growth was relatively slow. Today, however, WLAN technology is experiencing tremendous growth. The key reason for this growth is the increased bandwidth made possible by the IEEE 802.11 standard [10].

The IEEE initiated the 802.11 project in 1990 with a scope "to develop a MAC and PHY specification for wireless connectivity for fixed, portable, and moving stations within an area." In 1997, IEEE first approved the 802.11 international interoperability standards. In 1999, the IEEE ratified the 802.11a and the 802.11b wireless networking communication standards. The goal was to create a standards-based technology that span multiple physical encoding types, frequencies, and applications. The 802.11a standard uses OFDM to reduce interference. This technology uses the 5 GHz frequency spectrum and can process data at nearly up to 54 Mbps [10].

### 3.1.1 Physical Layer of 802.11a

The IEEE 802.11a standard specifies an OFDM PHY that splits an information signal across 52 separate subcarriers to provide transmission of data at a rate of 6, 9, 12, 18, 24, 36, 48, or 54 Mbps where the 6, 12, and 24 Mbps data rates are mandatory. Four of the subcarriers are pilot subcarriers that the system uses as a reference to disregard frequency or phase shifts of the signal during transmission where a pseudo binary sequence is sent through the pilot sub-channels to prevent the generation of spectral lines. the remaining 48 subcarriers provide separate wireless pathways for sending the information in a parallel fashion. The resulting subcarrier frequency spacing is 0.3125 MHz (for a 20 MHz bandwidth with 64 possible subcarrier frequency slots) [10].

Also in the 802.11a standard, the primary purpose of the OFDM PHY is to transmit MPDUs as directed by the 802.11 MAC layer. The OFDM PHY is divided into two elements: The PLCP and the PMD sublayers [10].

### 3.1.2 PPDU frame structure

The PSDU of the 802.11a is converted to a PPDU. The PSDU of the 802.11a is provided with a PLCP preamble and header to create the PPDU. Figure 3-1 shows the format for the PPDU including the OFDM PLCP preamble, OFDM PLCP header, PSDU, Tail bits, and Pad bits. The PLCP header contains the following fields: RATE, a reserved bit, LENGTH, an even parity bit, 6 Tail bits and the SERVICE field. In terms of modulation, the LENGTH, RATE, reserved bit, and parity bit (with 6 zero tail bits appended) constitute a separate single OFDM symbol denoted as SIGNAL. The SERVICE field of the PLCP header and the PSDU (with 6 zero tail bits and pad bits appended) denoted as DATA are transmitted at the data rate described in the RATE field and may constitute multiple OFDM symbols [10].

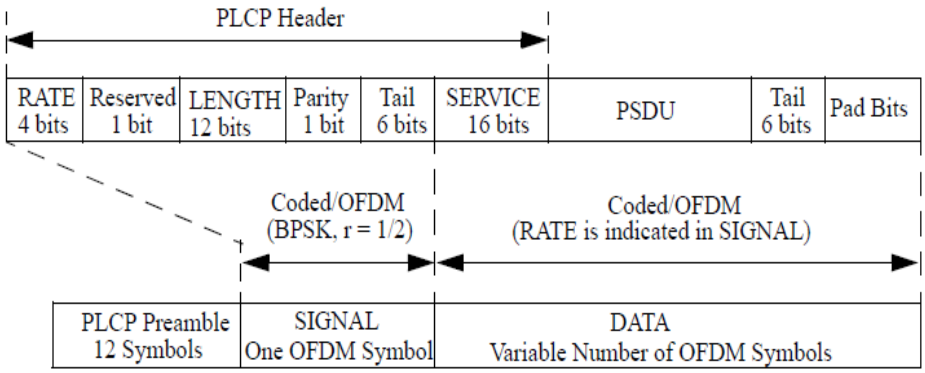


Figure 3-1: PPDU frame format

#### 3.1.2.1 PLCP preamble field

The PLCP preamble field acquire the incoming OFDM signal and train and synchronize the demodulator. The PLCP preamble is BPSK-OFDM modulated at 6 Mbps using convolutional encoding rate R=1/2 [10].

#### 3.1.2.2 SIGNAL field

These OFDM training symbols shall be followed by the SIGNAL field, which contains the RATE and the LENGTH fields of the TXVECTOR (PSDU). The RATE field conveys information about the type of modulation and the coding rate as used in the rest of the packet. The encoding of the SIGNAL single OFDM symbol

shall be performed with BPSK modulation of the subcarriers and using convolutional coding at R = ½ [10].

The encoding procedure, which includes convolutional encoding, interleaving, modulation mapping processes, pilot insertion, and OFDM modulation, follows the steps that used for transmission of data with BPSK-OFDM modulated at coding rate 1/2. The contents of the SIGNAL field are not scrambled [10].

The SIGNAL field shall be composed of 24 bits, as illustrated in Figure 5.2. The four bits 0 to 3 (R1-R4) shall encode the RATE. Bit 4 shall be reserved for future use. Bits 5–16 shall encode the LENGTH field of the TXVECTOR, with the LSB being transmitted first (the length of the PSDU, this length represent the number of octets in the PSDU). A continuation is a parity bit and 6 tail bits. The tail bits are set to "zeros" to facilitate a reliable and timely detection of the RATE and LENGTH fields [10].

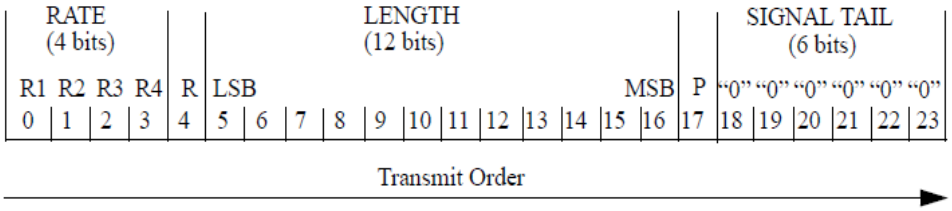


Figure 3-2: SIGNAL field bit assignment

In summary, the tail bits in the SIGNAL symbol enable decoding of the RATE and LENGTH fields immediately after the reception of the tail bits. The RATE and LENGTH fields are required for decoding the DATA part of the packet [10].

### 3.1.2.3 DATA field

The DATA field contains the SERVICE field, the PSDU, the TAIL bits, and the PAD bits, if needed. All bits in the DATA field are scrambled. The first 16 bits (7 null bits used for the scrambler initialization and 9 null bits reserved for future use) for the SERVICE field. A continuation is the PSDU. A continuation is a 6 tail bits and pad bits. The tail bits containing 0s are appended to the PPDU to ensure that the convolutional encoder returns to the zero state where this procedure improves the error probability of the convolutional decoder, which relies on future bits when decoding and which may be not be available past the end of the message and the pad bits are used as guards for the PPDU frame [10].

## 3.2 802.11a Transmitter PHY Block Diagram

As discussed in section 1.6.3, the graduation project team was able to implement the HDL codes for all transmitter blocks but the HDL codes were not synthesizable as each block needed more resources than the available resources by the FPGA. The main resource consumer was the memories that exist within each block for either operational or synchronization purposes.

This problem was solved by rewriting all memories HDL codes to match Xilinx Vivado HDL coding technique for memory without changing the memory interface within the block or the block functionality. Rewriting the codes allowed the Xilinx Vivado synthesizer to recognize the HDL code as RAM, either a block RAM or a dedicated RAM according to the memory size. In case of block RAM, a B-RAM resource is reserved. However, in case of dedicated RAM, a LUT resource is reserved and programed as RAM.

Transmitter of WI-FI consists of several blocks as shown in Figure 3-3. In the following sub-sections, each block of the chain is explained in more details _ illustrating its basic idea, showing its interfaces, connections, inputs & outputs & presenting its LUT utilization_.

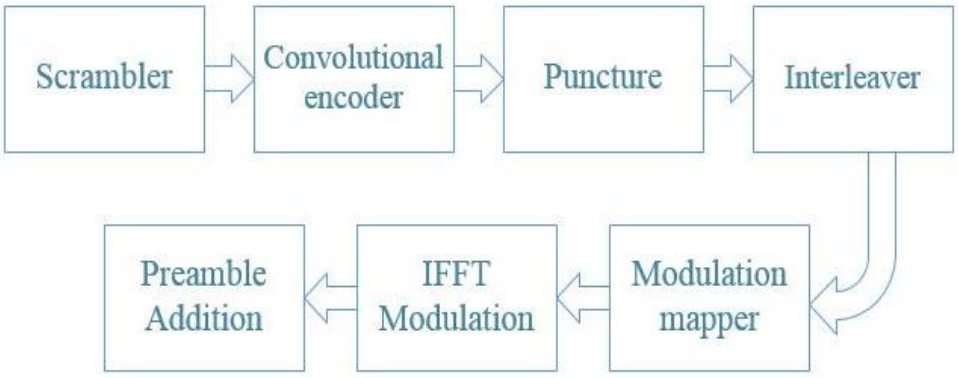


Figure 3-3: WI-FI Transmitter full chain blocks

### 3.2.1 Scrambler

Scrambler is used to randomize the service, PSDU, pad and data patterns to prevent long sequences of 1s or 0s to keep synchronization. The frame synchronous scrambler uses the generator polynomial S(x) as follows:

$$S(x) = x^7 + x^4 + 1$$

This generator polynomial S(x) can be represented as shown in Figure 3-4.

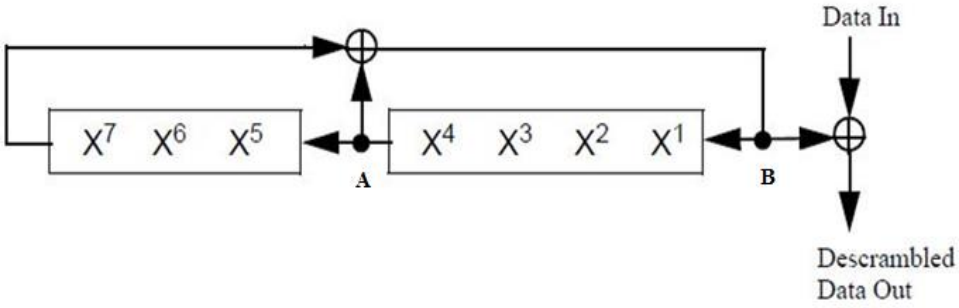


Figure 3-4: Data Scrambler

According to the initial state the scrambler will generate 127-bit sequence then it will return to its initial state. The same scrambler is used to scramble the transmitted data and descramble the received data The seven LSBs of the SERVICE field will be set to all zeros prior to scrambling to enable estimation of the initial state of the scrambler in the receiver.

The top controlled module of Scrambler is as shown in Figure 3-5, A detailed Scrambler module is as shown in Figure 3-6, and the pins description of the controlled module is shown in Table 3-1.

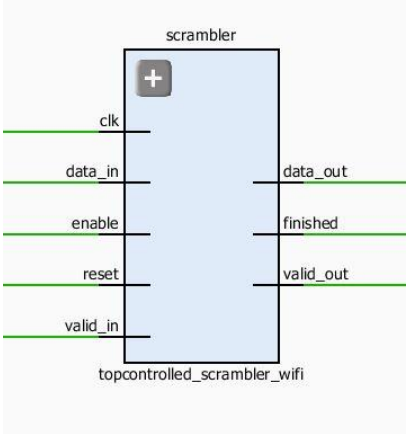


Figure 3-5: Top controlled Scrambler

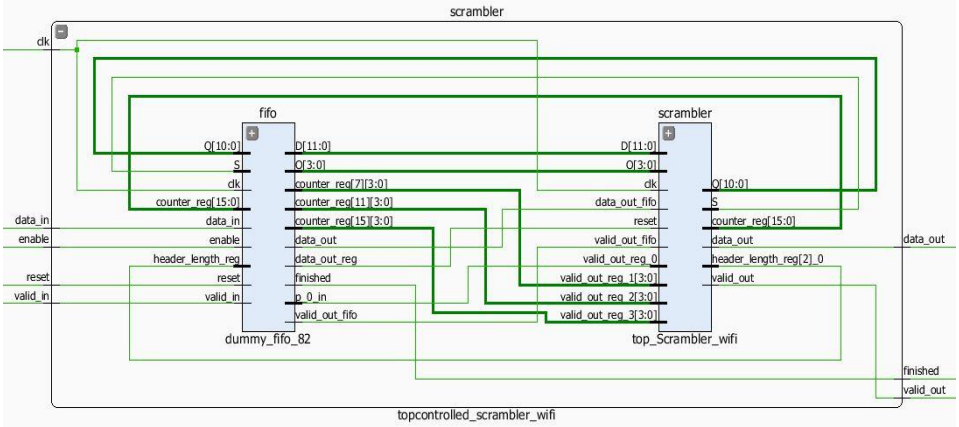


Figure 3-6: Top controlled Scrambler from inside

Table 3-1: pins description of Scrambler

| PIN | Description |
|---|---|
| data_in | The input bits to the block |
| data_out | The output data of the block |
| enable | This signal indicates that the next block (Encoder) is ready to have data |
| finished | This signal indicates that the Scrambler block is ready for a new frame |
| valid_in | This signal indicates that the current data_in is valid data |
| valid_out | This signal indicates that the current data_out is valid data |

Finally, the LUT utilization of the Scrambler block is shown in Figure 3-7.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 138 | 0 | 53200 | 0.26 |
| LUT as Logic | 136 | 0 | 53200 | 0.26 |
| LUT as Memory | 2 | 0 | 17400 | 0.01 |
| LUT as Distributed RAM | 2 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 80 | 0 | 106400 | 0.08 |
| Register as Flip Flop | 80 | 0 | 106400 | 0.08 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 3-7: Scrambler LUT utilization

### 3.2.2 Convolutional Encoder

The DATA field, composed of SERVICE, PSDU, tail, and pad parts, shall be coded with a convolutional encoder of coding rate R = 1/2, 2/3, or 3/4, corresponding to the desired data rate. The convolutional encoder shall use the industry-standard generator polynomials, g0 = 1338 and g1 = 1718, of rate R = 1/2, as shown in Figure 3-8. The bit denoted as "A" shall be output from the encoder before the bit denoted as "B." Higher rates are derived from it by employing "puncturing." Puncturing is a procedure for omitting some of the encoded bits in the transmitter (thus reducing the number of transmitted bits and increasing the coding rate) and inserting a dummy "zero" metric into the convolutional decoder on the receive side in place of the omitted bits. The encoder is followed by parallel to serial block to transmit the encoded bits to the puncture [2].

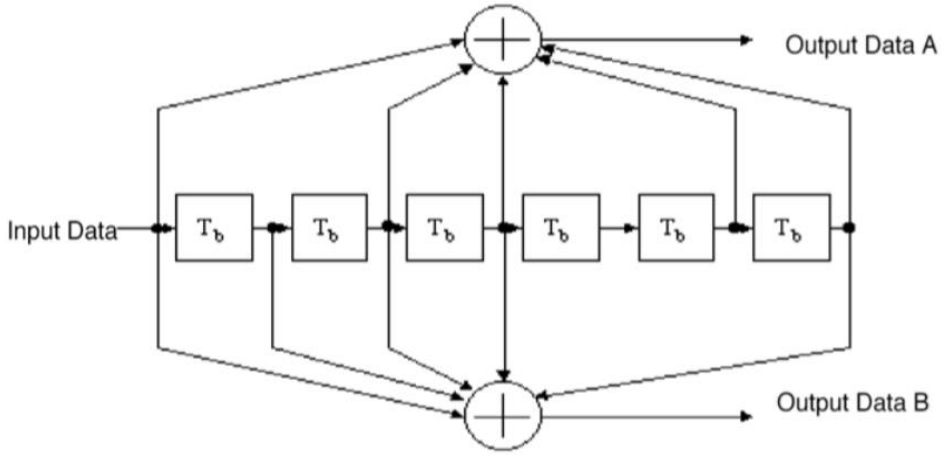


Figure 3-8: Convolutional Encoder (K=7)

The top controlled module of Encoder is as shown in Figure 3-9, A detailed Encoder module is as shown in Figure 3-10, and the pins description of the controlled module is shown in Table 3-2.

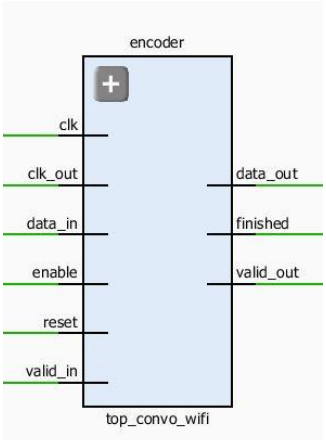


Figure 3-9: Top controlled Encoder

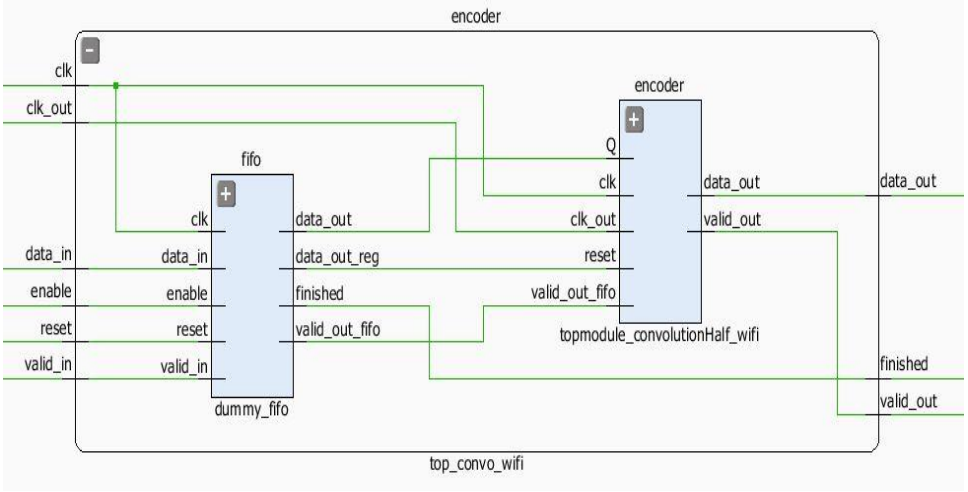


Figure 3-10: Top controlled Encoder from inside

Table 3-2: pins description of Encoder

| PIN | Description |
|---|---|
| clk_out | Clock of the parallel output |
| data_in | The input bits to the block |
| data_out | The output bits of the block |
| enable | This signal indicates that the next block (Puncture) is ready to have data |
| finished | This signal indicates that the Encoder block is ready for the new frame |
| valid_in | This signal indicates that current data_in is valid data |
| valid_out | This signal indicates that current data_out is valid data |

Finally, the LUT utilization of the Encoder block is shown in Figure 3-11.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 73 | 0 | 53200 | 0.14 |
| LUT as Logic | 71 | 0 | 53200 | 0.13 |
| LUT as Memory | 2 | 0 | 17400 | 0.01 |
| LUT as Distributed RAM | 2 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 50 | 0 | 106400 | 0.05 |
| Register as Flip Flop | 50 | 0 | 106400 | 0.05 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 3-11: Encoder LUT utilization

### 3.2.3 Puncture

If the system could only change the data rate by adjusting the constellation size, and not the code rate, a very large number of different rates would be difficult to achieve as the number of constellations and the number of points in the largest constellation would grow very quickly. Another solution would be to implement several different convolutional encoders with different rates and change both the convolutional code rate and constellation. However, this approach has problems in the receiver that would have to implement several different decoders for all the codes used. Puncturing is a very useful technique to generate additional rates from a single convolutional code.

The basic idea behind puncturing is to not transmit some of the output bits from the convolutional encoder, thus increasing the rate of the code and inserting a dummy zero metric into the convolutional decoder on the receive side in place of the omitted bits, hence only one encoder/decoder pair is needed to generate several different code rates. The puncture pattern is specified by the Puncture vector parameter in the mask. The puncture vector is a binary column vector. A 1 indicates that the bit in the corresponding position of the input vector is sent to the output vector, while a 0 indicates that the bit is removed. There are two types of punctures in WI-FI standard: (2/3) and (3/4) according to the data rate.

The top controlled module of Puncture is as shown in Figure 3-12, and the pins description of the controlled module is shown in Table 3-3.

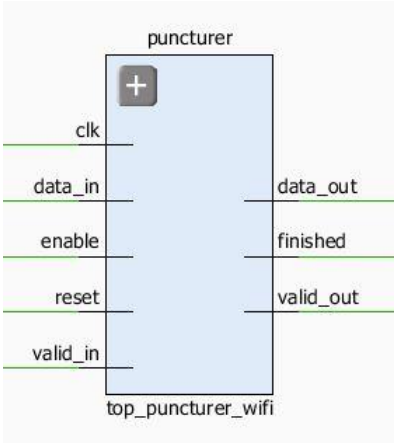


Figure 3-12: Top controlled Puncture

Table 3-3: pins description of Puncture

| PIN | Description |
|---|---|
| data_in | The input bits to the block |
| data_out | The output bits of the block |
| enable | This signal indicates that the next block (Interleaver) is ready to have data |
| finished | This signal indicates that the Puncture block is ready for the new frame |
| valid_in | This signal indicates that current data_in is valid data |
| valid_out | This signal indicates that current data_out is valid data |

Finally, the LUT utilization of the Puncture block is shown in Figure 3-13.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 165 | 0 | 53200 | 0.31 |
| LUT as Logic | 163 | 0 | 53200 | 0.31 |
| LUT as Memory | 2 | 0 | 17400 | 0.01 |
| LUT as Distributed RAM | 2 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 109 | 0 | 106400 | 0.10 |
| Register as Flip Flop | 109 | 0 | 106400 | 0.10 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 3-13: Puncture LUT utilization

### 3.2.4 Interleaver

All encoded data bits shall be interleaved by a block Interleaver with a block size corresponding to the number of bits in a single OFDM symbol. The Interleaver is defined by a two-step permutation. The first permutation ensures that adjacent coded bits are mapped onto nonadjacent subcarriers. The second ensures that adjacent coded bits are mapped alternately onto less and more significant bits of the constellation and, thereby, long runs of low reliability (LSB) bits are avoided.

The top controlled module of Interleaver is as shown in Figure 3-14, and the pins description of the controlled module is shown in Table 3-4.

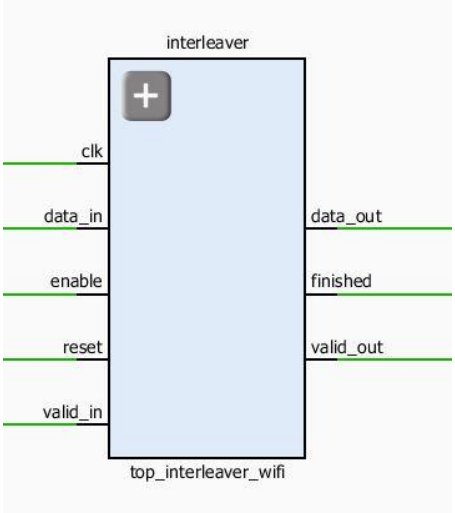


Figure 3-14: Top controlled Interleaver

Table 3-4: pins description of Interleaver

| PIN | Description |
|---|---|
| data_in | The input bits to the block |
| data_out | The output bits of the block |
| enable | This signal indicates that the next block (Mapper) is ready to have data |
| finished | This signal indicates that the Interleaver block is ready to have a new frame |
| valid_in | This signal indicates that current data_in is valid data |
| valid_out | This signal indicates that current data_out is valid data |

Finally, the LUT utilization of the Interleaver block is shown in Figure 3-15.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 176 | 0 | 53200 | 0.33 |
| LUT as Logic | 176 | 0 | 53200 | 0.33 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 124 | 0 | 106400 | 0.12 |
| Register as Flip Flop | 124 | 0 | 106400 | 0.12 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 3-15: Interleaver LUT utilization

### 3.2.5 Modulation Mapper

Modulation is the process by which information (e.g. bit stream) is transformed into sinusoidal waveform. A sinusoidal wave has three features those can be changed _phase, frequency and amplitude_ according to the given information and to the used modulation technique. In 802.11a Phase Shift Keying (BPSK, QPSK) and Quadrature Amplitude Modulation (16-QAM, 64-QAM) modulation techniques are used according to the desired data rate as described in the following equation: $d = (I + j\,Q) * K_{mod}$ where $K_{mod}$ is the normalization factor and is used in to achieve the same average power for all mappings. It depends on the base modulation mode where for BPSK, $K_{mod} = 1$, for QPSK, $K_{mod} = 1/\sqrt{2}$ , for 16-QAM, $K_{mod} = 1/\sqrt{10}$, for 64-QAM, $K_{mod} = 1/\sqrt{40}$.

Every modulation mode has a modulation specified in the standard as shown in Figure 3-16.

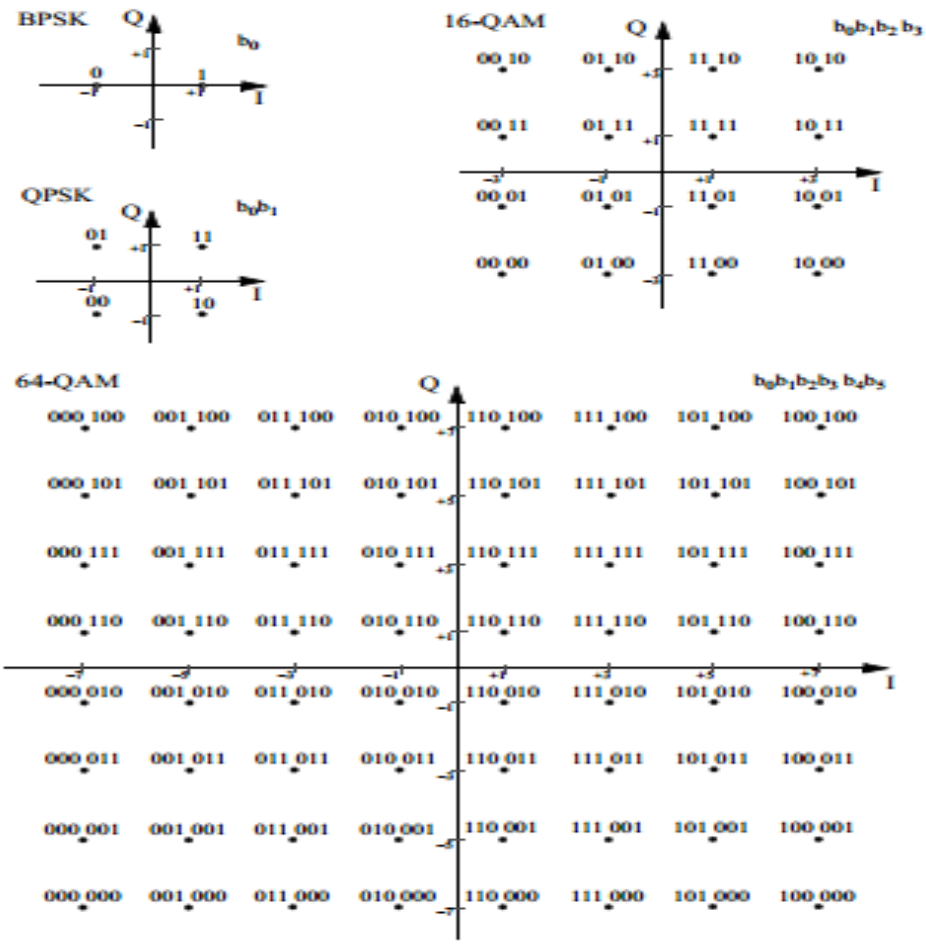


Figure 3-16: Modulation constellations for BPSK, QPSK, 16-QAM, and 64-QAM

The top controlled module of Mapper is as shown in Figure 3-17, A detailed Mapper module is as shown in Figure 3-18, and the pins description of the controlled module is shown in Table 3-5.

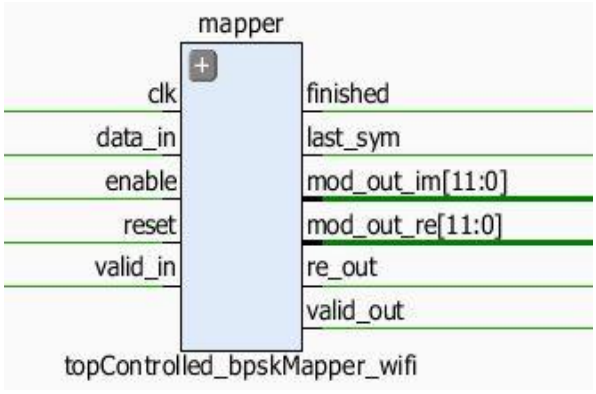


Figure 3-17: Top controlled Mapper

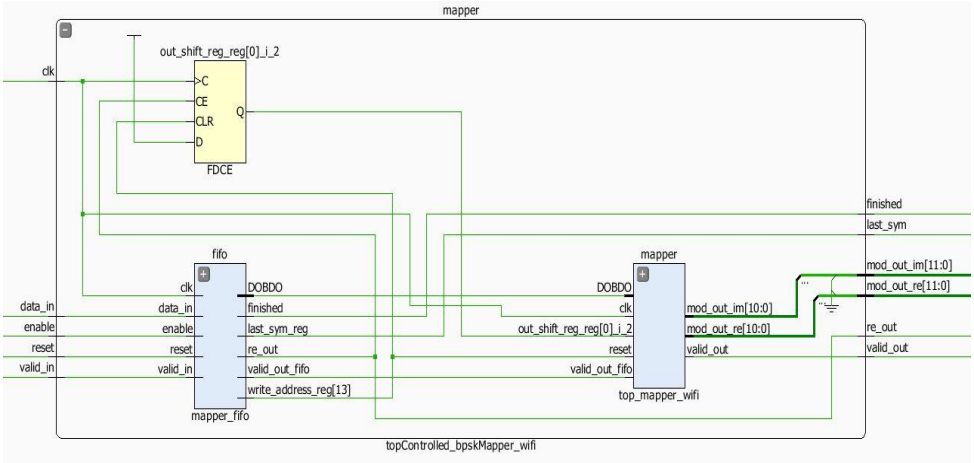


Figure 3-18: Top controlled Mapper from inside

Table 3-5: pins description of Mapper

| PIN | Description |
|---|---|
| data_in | The input bits to the block |
| enable | This signal indicates that the next block (IFFT Modulation) is ready to have data |
| finished | This signal indicates that the Mapper block is ready to have a new frame |
| mod_out_im | The modulated imaginary part of the input |
| mod_out_Re | The modulated real part of the input |
| valid_in | This signal indicates that current data_in is valid data |
| valid_out | This signal indicates that current data_out is valid data |
| last_sym | This signal indicates that the Mapper marks to the IFFT that these patch of symbols are the last patch to be processed |

Finally, the LUT utilization of the Mapper block is shown in Figure 3-19.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 83 | 0 | 53200 | 0.16 |
| LUT as Logic | 83 | 0 | 53200 | 0.16 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 49 | 0 | 106400 | 0.05 |
| Register as Flip Flop | 49 | 0 | 106400 | 0.05 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 3-19: Mapper LUT utilization

### 3.2.6 IFFT Modulation

WIFI uses orthogonal frequency division multiplexing for modulation, An OFDM signal consists of a number of closely spaced modulated carriers as shown in Figure 3-20 , those carriers are orthogonal so the receiver could demodulate them, OFDM systems are very sensitive to frequency offset and ISI because any error in the received signal affects all carriers and all data so a guard interval is used between OFDM symbols, In this guard signal we insert a cyclic prefix of the symbol to compensate for any synchronization problems with in the receiver [11].

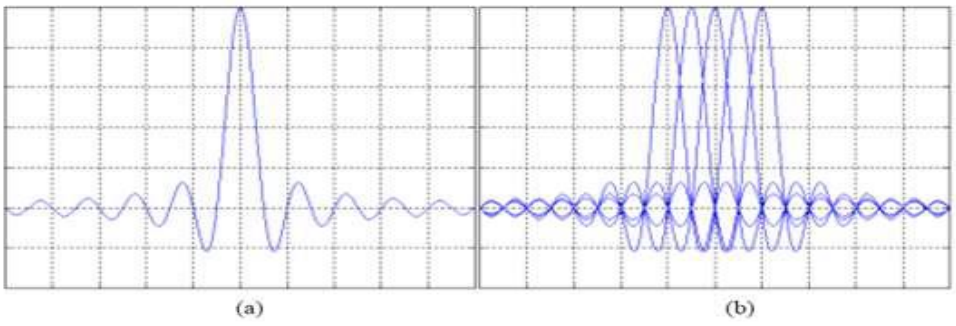


Figure 3-20: (a) Spectrum of a single subcarrier of the OFDM signal,
(b) Spectrum of the OFDM signal

The important parameters for the OFDM modulation system are the number of subcarriers used within the bandwidth, the cyclic prefix and where to insert pilot signals. Inverse fast Fourier transform is used for the modulation operation, as specified by the IEEE 802.11a, 64-point IFFT is used with symbol duration of 4 us in the 20 MHz operation of the standard. The symbol time consists of a 3.2 us

symbol and 0.8 us for the cyclic prefix, the timing of the OFDM frame is as shown in Figure 3-21 [11].

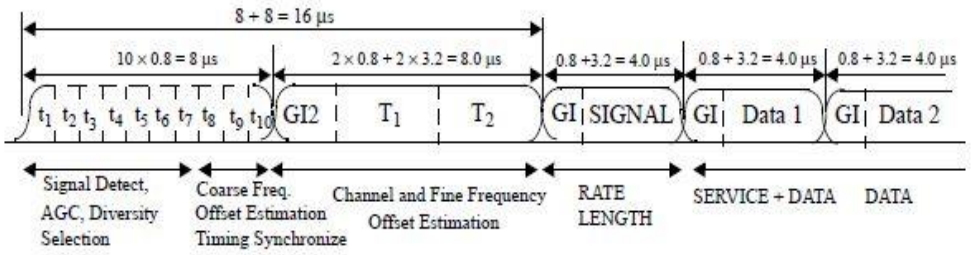


Figure 3-21: OFDM training structure

The single OFDM symbol contains 48 data symbols from the mapper, contains 4 pilot symbols, 11 null symbol and null input at DC, this mapping is shown in the below function where k is the logical subcarrier number and M(k) is the frequency offset index, the frequency offset index mapping to the IFFT inputs is shown in Figure 3-22 [11].

$$M(k) = \begin{cases} k - 26 & 0 \le k \le 4 \\ k - 25 & 5 \le k \le 17 \\ k - 24 & 18 \le k \le 23 \\ k - 23 & 24 \le k \le 29 \\ k - 22 & 30 \le k \le 42 \\ k - 21 & 43 \le k \le 47 \end{cases}$$

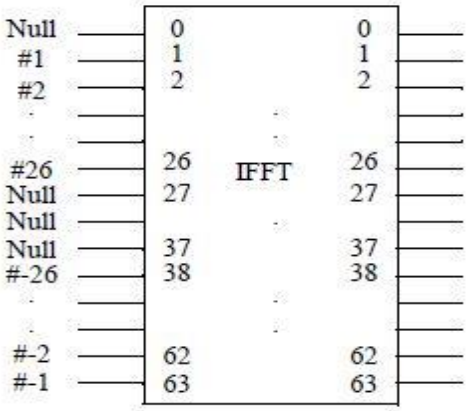


Figure 3-22: frequency offset index function & inputs and outputs of the IFFT

Pilots are inserted at subcarriers -21, -7, 7, 21. So, the final mapping of the 64 subcarrier is as shown in Figure 3-23 [11].

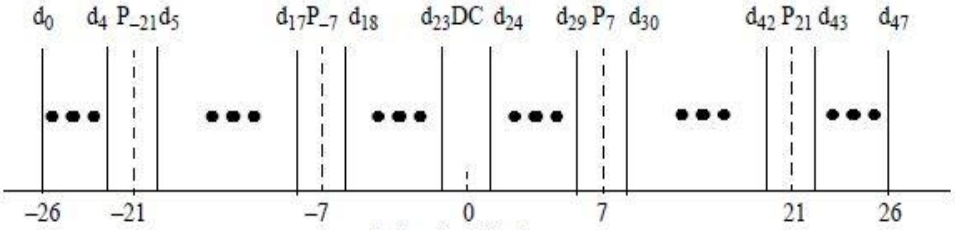


Figure 3-23: Final 64 subcarrier mapping

The hardware circuit implementation needs an IFFT circuit, we used the Xilinx LogiCORE IP Fast Fourier Transform v7.1, and the IP has many options we used

the pipelined streaming I/O to ensure continuous output to comply with the standard requirements.

The top controlled module of IFFT Modulation is as shown in Figure 3-24, and the pins description of the controlled module is shown in Table 2-6.

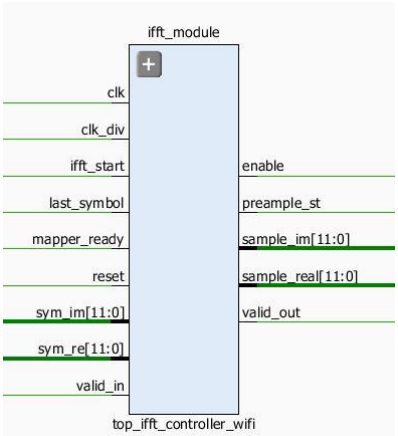


Figure 3-24: Top controlled IFFT Modulation

Table 3-6: pins description of IFFT Modulation

| PIN | Description |
|---|---|
| clk_div | clock of the output |
| ifft_start | This signal indicates that the IFFT is ready to receive data |
| last_symbol | This signal indicates that the Mapper marks to the IFFT that these patch of symbols are the last patch to be processed |
| mapper_ready | This signal indicates that the Mapper starts to input symbols to its pipeline |
| sample_im | Imaginary output data of the block |
| sample_re | Real output data of the block |
| sym_im | Imaginary input data to the block |
| sym_re | Real input data to the block |
| valid_in | This signal indicates that current data_in is valid data |
| valid_out | This signal indicates that current data_out is valid data |

Finally, the LUT utilization of the IFFT Modulation is shown in Figure 3-25.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 1886 | 0 | 53200 | 3.55 |
| LUT as Logic | 1292 | 0 | 53200 | 2.43 |
| LUT as Memory | 594 | 0 | 17400 | 3.41 |
| LUT as Distributed RAM | 64 | 0 | | |
| LUT as Shift Register | 530 | 0 | | |
| Slice Registers | 1810 | 0 | 106400 | 1.70 |
| Register as Flip Flop | 1810 | 0 | 106400 | 1.70 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 27 | 0 | 26600 | 0.10 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 3-25: IFFT Modulation LUT utilization

### 3.2.7 Preamble

In WI-FI 802.11a, The PLCP Preamble field is used for synchronization. It consists of 10 short symbols and two long symbols that are shown in Figure 3-21. The timings described in this sub-clause and shown in Figure 3-21 are for 20 MHz channel spacing. They are doubled for half-clocked (i.e., 10 MHz) channel spacing and are quadrupled for quarter-clocked (i.e., 5 MHz) channel spacing.

Figure 3-21 shows the OFDM training structure (PLCP preamble), where t1 to t10 denotes short training symbols and T1 and T2 denote long training symbols. The total training length is 16μs. The dashed boundaries in the figure denote repetitions due to the periodicity of the inverse Fourier transform. The PLCP preamble shall be transmitted using an OFDM modulated fixed waveform.

The top controlled module of Preamble is as shown in Figure 3-26, A detailed Preamble module is as shown in Figure 3-27, and the pins description of the controlled module is shown in Table 2-7.

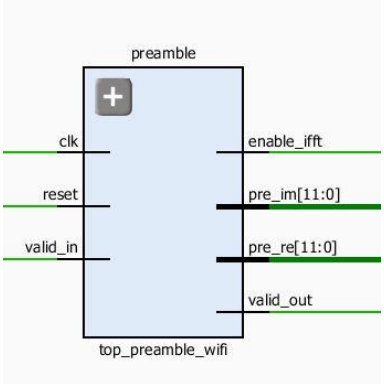


Figure 3-26: Top controlled Preamble

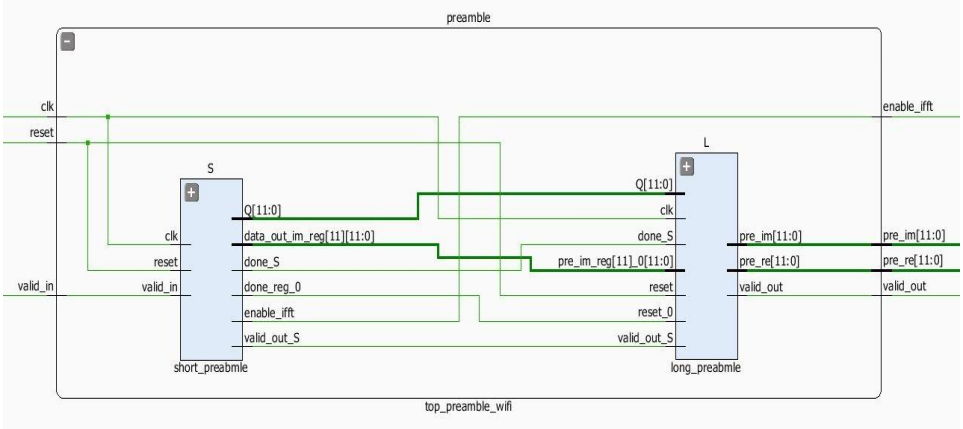


Figure 3-27: Top controlled Preamble from inside

Table 3-7: pins description of Preamble

| PIN | Description |
|---|---|
| enable_ifft | This signal used to enable the IFFT Modulation block in order to output data |
| pre_im | The imaginary part of the Preamble |
| pre_re | The real part of the Preamble |
| valid_in | The signal indicates that current data_in is valid data |
| valid_out | This signal indicates that current data_out is valid data |

Finally, the LUT utilization of the Preamble block is shown in Figure 3-28.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 123 | 0 | 53200 | 0.23 |
| LUT as Logic | 123 | 0 | 53200 | 0.23 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 57 | 0 | 106400 | 0.05 |
| Register as Flip Flop | 57 | 0 | 106400 | 0.05 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 1 | 0 | 26600 | <0.01 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 3-28: Preamble LUT utilization

The LUT utilization of the WI-FI transmitter full chain is shown in Figure 3-29.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 2559 | 0 | 53200 | 4.81 |
| LUT as Logic | 1961 | 0 | 53200 | 3.69 |
| LUT as Memory | 598 | 0 | 17400 | 3.44 |
| LUT as Distributed RAM | 68 | 0 | | |
| LUT as Shift Register | 530 | 0 | | |
| Slice Registers | 2207 | 0 | 106400 | 2.07 |
| Register as Flip Flop | 2207 | 0 | 106400 | 2.07 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 28 | 0 | 26600 | 0.11 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 3-29: WI-FI transmitter full chain LUT utilization

## 3.3 802.11a Receiver PHY Block Diagram

The main target of the receiver is to retrieve the same data send before transmitter so it consists of the blocks shown in Figure 3-30. The same procedure is performed in the receiver blocks as in the transmitter blocks where in the following sub-sections, each block of the chain is explained in more details _ illustrating its basic idea, showing its interfaces, connections, inputs & outputs & presenting its LUT utilization_.

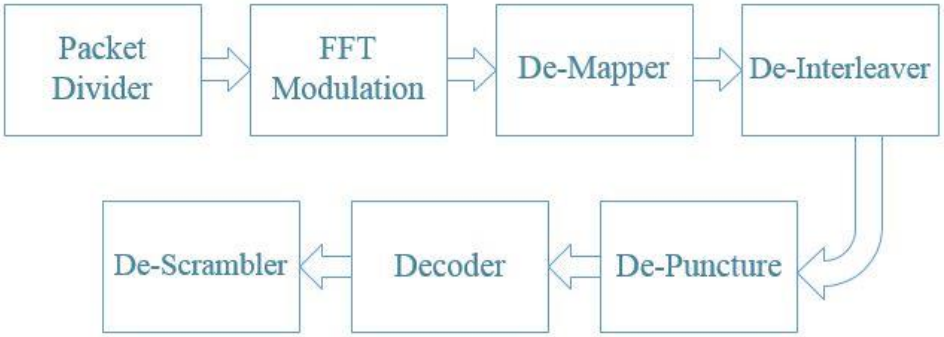


Figure 3-30: WI-FI Receiver full chain blocks

### 3.3.1 Packet Divider

It is the first block of the receiver that receives the real and imaginary data of the channel which came in the form of 12 bits divided to 9 bits representing the fraction part and 3 bits representing the real part. The main target of the block is to receive these data symbols, store them, remove preamble from them and deliver the rest to the next block (FFT Modulation).

The top controlled module of Packet Divider is as shown in Figure 3-31, and the pins description of the controlled module is shown in Table 3-8.

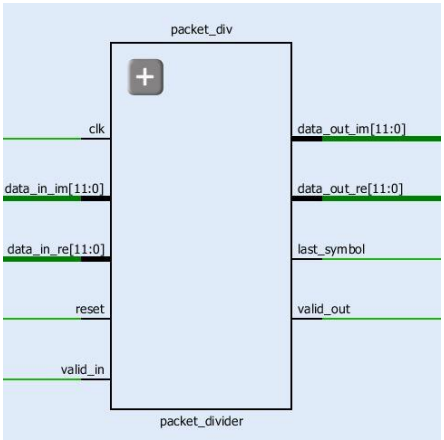


Figure 3-31: Top controlled Packet Divider

Table 3-8: pins description of Packet Divider

| Pin | Description |
|---|---|
| data_in_im | Imaginary input data to the block |
| data_in_re | Real input data to the block |
| data_out_im | Imaginary output data of the block |
| data_out_re | Real output data of the block |
| last_symbol | This signal indicates that Packet Divider marks to the FFT that these patch of symbols are the last patch to be processed |
| valid_in | This signal indicates that current data_in is valid |
| valid_out | This signal indicates that current data_out is valid |

Finally, the LUT utilization of the Packet Divider block is shown in Figure 3-32.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 27 | 0 | 53200 | 0.05 |
| LUT as Logic | 27 | 0 | 53200 | 0.05 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 45 | 0 | 106400 | 0.04 |
| Register as Flip Flop | 45 | 0 | 106400 | 0.04 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 3-32: Packet Divider LUT utilization

### 3.3.2 FFT Modulation

As specified in section 3.2.6, WIFI uses orthogonal frequency division multiplexing for modulation where the hardware circuit implementation needs an FFT circuit, we used the Xilinx LogiCORE IP Fast Fourier Transform v7.1 as in the transmitter.

The first challenge is to make the data received from the Packet Divider block ready to enter the FFT core without the cyclic prefix, and to control the pipelining process such that the latency reduces as much as possible, so a FFT controller is needed to control the whole process.

The FFT controller contains the FFT core and four RAMs with size 64 x 12 where two RAMs are used for the real part & the other two RAMs are used for the imaginary part and concerning the RAM size, the FFT core receives 64 inputs with length of 12 bits. The RAM purpose is to store the input data without cyclic prefix

and maintain the process of pipelining as when a RAM is getting an input, the other one delivers its data to the FFT core. This process is a continuous one till a last symbol flag is raised from the packet divider which is an indication that there is no more data to process on. While the reading and writing processes of the RAMs are being on, the FFT core is producing its output to the next block (De-Mapper).

The top controlled module of FFT Modulation is as shown in Figure 3-33, and the pins description of the controlled module is shown in Table 3-9.

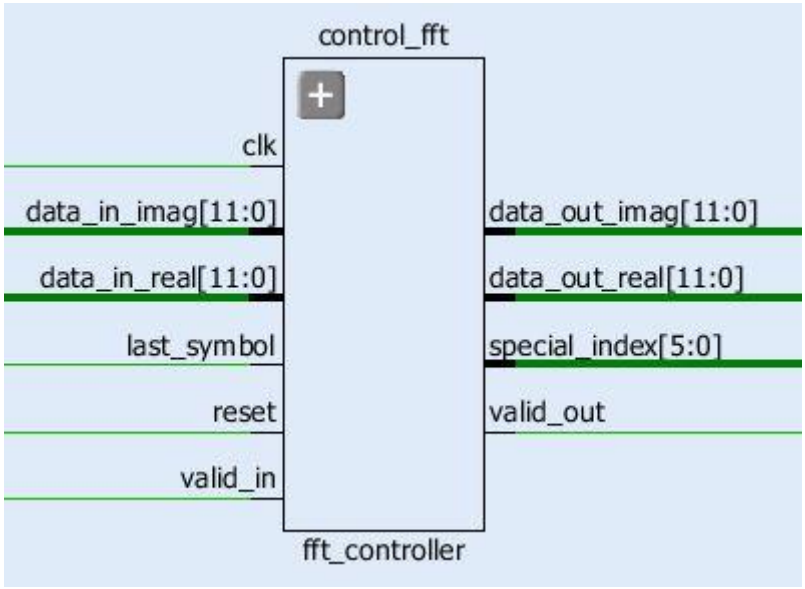


Figure 3-33: Top controlled FFT Modulation

Table 3-9: pins description of FFT Modulation

| Pin | Description |
|---|---|
| data_in_imag | Imaginary input data to the block |
| data_in_real | Real input data to the block |
| data_out_imag | Real output data of the block |
| data_out_real | Imaginary output data of the block |
| last_symbol | This signal indicates that Packet Divider marks to the FFT that these patch of symbols are the last patch to be processed |
| special_index | This signal used in the De-Mapper to define the pilots and nulls |
| valid_in | This signal indicates that current data_in is valid |
| valid_out | This signal indicates that current data_out is valid |

Finally, the LUT utilization of the FFT Modulation block is shown in Figure 3-34.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 2072 | 0 | 53200 | 3.89 |
| LUT as Logic | 1472 | 0 | 53200 | 2.77 |
| LUT as Memory | 600 | 0 | 17400 | 3.45 |
| LUT as Distributed RAM | 64 | 0 | | |
| LUT as Shift Register | 536 | 0 | | |
| Slice Registers | 2036 | 0 | 106400 | 1.91 |
| Register as Flip Flop | 2036 | 0 | 106400 | 1.91 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 26 | 0 | 26600 | 0.10 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 3-34: FFT Modulation LUT utilization

### 3.3.3 De-Mapper

It receives the real and imaginary data from the FFT Modulation block which came in the form of 12 bits divide to 9 bits represent the fraction part and 3 bits represent the real part. The main target of the block is to receive these data symbols, specify the decision region and convert these symbols to a stream of bits.

As specified in section 3.2.5, 802.11a Phase Shift Keying (BPSK, QPSK) and Quadrature Amplitude Modulation (16-QAM, 64-QAM) modulation techniques are used where the decision regions are shown in Figure  It's to be noted the .35-364-QAM  modulation technique isn't included.

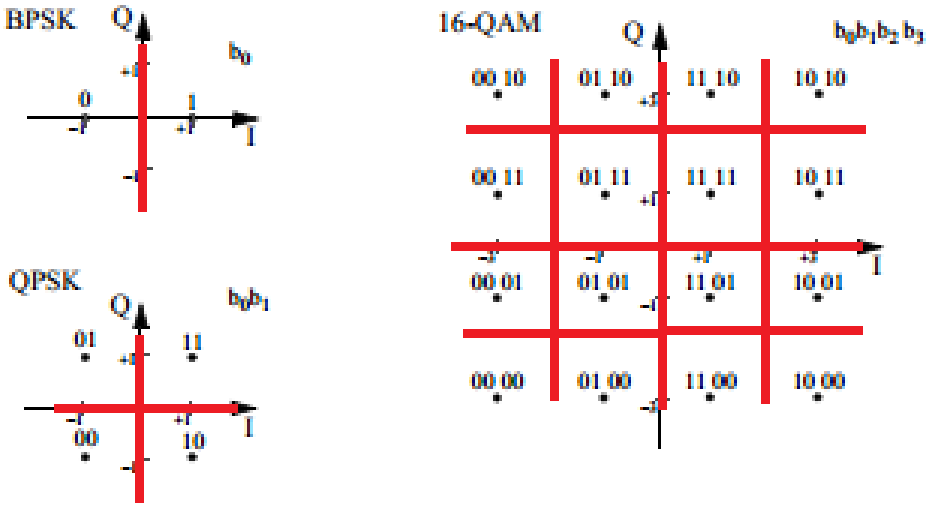


Figure 3-35: Decision regions of De-Mapper

The block design is a bit more different than any usual De-Mapper design where it contains three main blocks. So, a controller is required to organize the signaling flow.

The first block is a stack controller which contains four stacks where two are used for real part and the other two for imaginary part to maintain the pipelining process (De-mapping while the FFT core is working). The reason for using stack structure not RAM is to cancel the transmitter effect where in the Mapper, the input data was reversed (stored up-down then read down-up). So, a stack is used with size 48 x 12 as the output of the FFT core is in the form of 64 data blocks and 18 sample aren't needed in the De-Mapper which are 4 pilots and 14 null.

The second block is the main De-Mapper block which decode the symbols into bits through the decision regions according to the modulation scheme used in the transmitter.

The third block is a large RAM used to store the whole data as the next block (De-Interleaver) needs all the frame data to be ready to work properly.

Three clocks are being used in the de-mapping process to maintain the pipelining process.

1. The first clock is the normal FFT clock (system clock divided by 10) for BPSK de-mapping.
2. The second clock is the output clock (system clock divided by 4) to speed up the de-mapping process in case of the modulation scheme QPSK where the De-Mapper needs more time than in the case of BPSK to decode the symbols correctly.
3. The third clock is the fastest clock in the system which is the decoder clock (system clock divided by 2) to speed up the de-mapping process in case of 16 QAM.

The top controlled module of De-Mapper is as shown in Figure 3-36, A detailed De-Mapper module is as shown in Figure 3-37, and the pins description of the controlled module is shown in Table 3-10.

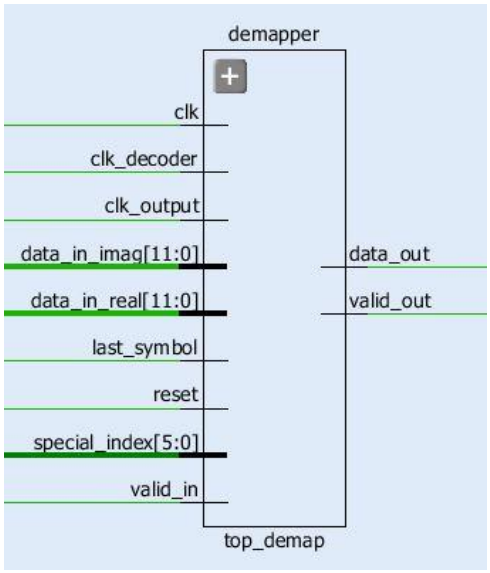


Figure 3-36: Top controlled De-Mapper

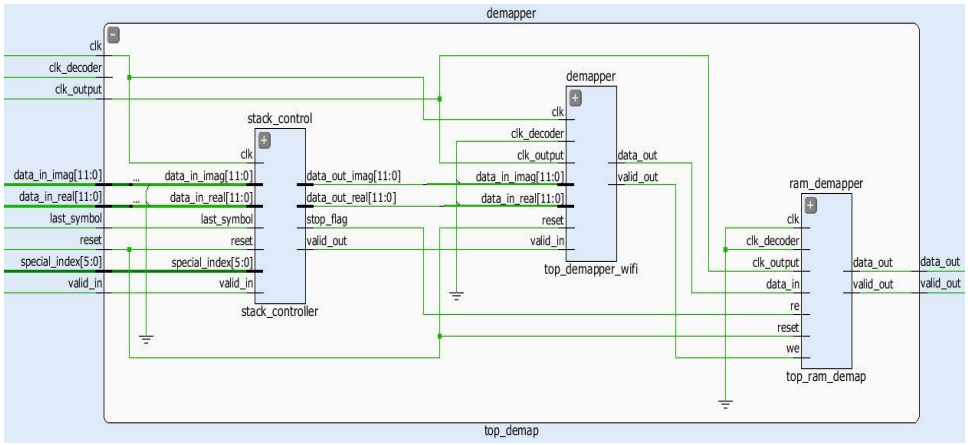


Figure 3-37: Top controlled De-Mapper from inside

Table 3-10: pins description of De-Mapper

| Pin | Description |
|---|---|
| clk_decoder | Used for de-mapping in case of 16 QAM modulation scheme |
| clk_output | Used for de-mapping in case of QPSK modulation scheme |
| data_in_imag | Imaginary input data to the block |
| data_in_real | Real input data to the block |
| last_symbol | This signal indicates that Packet Divider marks to the FFT that these patch of symbols are the last patch to be processed |
| special_index | This signal used in the De-Mapper to define the pilots and nulls |
| valid_in | This signal indicates that current data_in is valid |
| data_out | The output data of the block |
| valid_out | This signal indicates that current data_out is valid |

Finally, the LUT utilization of the De-Mapper block is shown in Figure 3-38.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 342 | 0 | 53200 | 0.64 |
| LUT as Logic | 326 | 0 | 53200 | 0.61 |
| LUT as Memory | 16 | 0 | 17400 | 0.09 |
| LUT as Distributed RAM | 16 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 210 | 0 | 106400 | 0.20 |
| Register as Flip Flop | 210 | 0 | 106400 | 0.20 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 3-38: De-Mapper LUT utilization

### 3.3.4 De-Interleaver

The de-Interleaver, which performs the inverse relation to the Interleaver, is also defined by two permutations. Here the index of the original received bit before the first permutation shall be denoted by (j); (d) shall be the index after the first and before the second permutation; and (e) shall be the index after the second permutation, just prior to delivering the coded bits to the convolutional (Viterbi) decoder (if de-puncture isn't used).

The first permutation is defined by the rule:

$$\mathrm{d} = \mathrm{s} * \mathrm{Floor}\left(\frac{\mathrm{j}}{\mathrm{s}}\right) + \left(j + Floor\left(16 * \frac{j}{N_{CBPS}}\right)\right) mod\ s \qquad j = 0,1 \ldots, N_{CBPS} - 1$$

The second permutation is defined by the rule:

$$\mathrm{e} = 16 * \mathrm{d} - (N_{CBPS} - 1) * Floor\left(16 * \frac{d}{N_{CBPS}}\right) \qquad d = 0,1 \ldots, N_{CBPS} - 1$$

These permutations represent the inverse equations to the permutation equations in the Interleaver of the transmitter.

The value of s is determined by the number of coded bits per subcarrier, $N_{BPSC}$, according to:

$$s = \max\left(\frac{N_{BPSC}}{2}, 1\right)$$

The top controlled module of De-Interleaver is as shown in Figure 3-39, and the pins description of the controlled module is shown in Table 3-11.

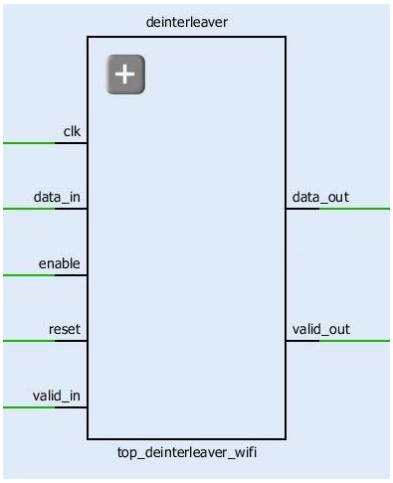


Figure 3-39: Top controlled De-Interleaver

Table 3-11: pins description of De-Interleaver

| PIN | Description |
|---|---|
| data_in | The input bits to the block |
| data_out | The output bits of the block |
| enable | This signal indicates that the next block (De-puncture) is ready to have data |
| finished | This signal indicates that the De-Interleaver is ready to have a new frame |
| valid_in | This signal indicates that current data_in is valid data |
| valid_out | This signal indicates that current data_out is valid data |

Finally, the LUT utilization of the De-Interleaver block is shown in Figure 3-40.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 346 | 0 | 53200 | 0.65 |
| LUT as Logic | 346 | 0 | 53200 | 0.65 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 127 | 0 | 106400 | 0.12 |
| Register as Flip Flop | 127 | 0 | 106400 | 0.12 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 3-40: De-Interleaver LUT utilization

### 3.3.5 De-Puncture

De-Puncture is the reverse block of puncture. De-Puncture adds dummy bits in the position of removed bits by puncture. The positions of removed bits are determined in the standard in the puncture vector which is a binary column vector as explained in section 3.2.3. Figure 3-41 shows the procedure of puncture and de-puncture of rate 3/4. Figure 3-42 shows the procedure of puncture and de-puncture of rate 2/3.

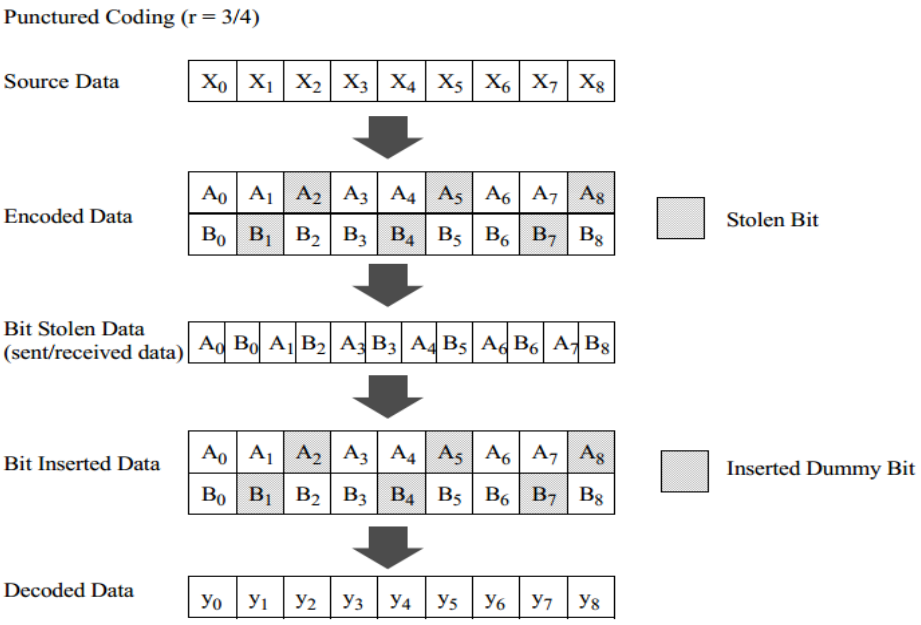


Figure 3-41: De-Puncture 3/4 rate procedure

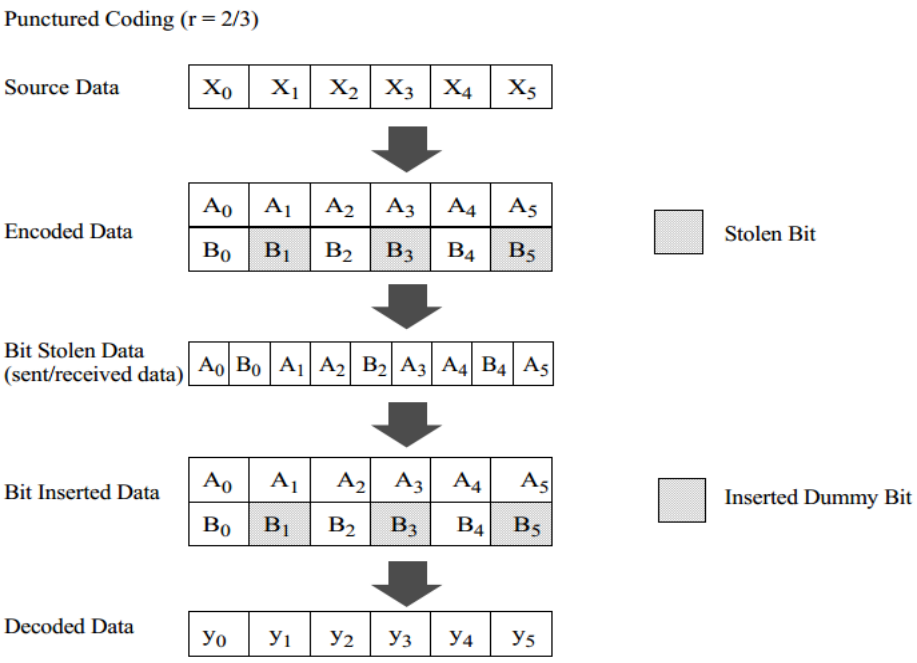


Figure 3-42: De-Puncture 2/3 rate procedure

The top controlled module of De-Puncture is as shown in Figure 3-43, and the pins description of the controlled module is shown in Table 3-12.

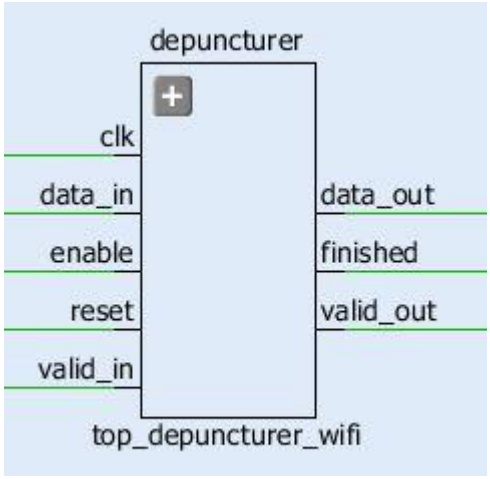


Figure 3-43: Top controlled De-Puncture

Table 3-12: pins description of De-Puncture

| Pin | Description |
|---|---|
| data_in | The input data to the block |
| data_out | The output data of the block |
| enable | This signal indicates that the next block (Decoder) is ready to have data |
| finished | This signal indicates that the De-puncture is ready to have data |
| valid_in | This signal indicates that current data_in is valid |
| valid_out | This signal indicates that current data_out is valid |

Finally, the LUT utilization of the De-Puncture block is shown in Figure 3-44.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 165 | 0 | 53200 | 0.31 |
| LUT as Logic | 163 | 0 | 53200 | 0.31 |
| LUT as Memory | 2 | 0 | 17400 | 0.01 |
| LUT as Distributed RAM | 2 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 109 | 0 | 106400 | 0.10 |
| Register as Flip Flop | 109 | 0 | 106400 | 0.10 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 3-44: De-Puncture LUT utilization

### 3.3.6 Viterbi Decoder

Viterbi Decoder is the reverse block of the convolutional encoder. The block design is the same as that described in section 2.3.5, the only differences exist in that the used convolutional encoder has K=7 so as the Viterbi decoder here which decreases the number of the states to 64 other than 256, also in this design there is no tail bit removing.

The top controlled module of Viterbi decoder is as shown in Figure 3-45, and the pins description of the controlled module is shown in Table 3-13.

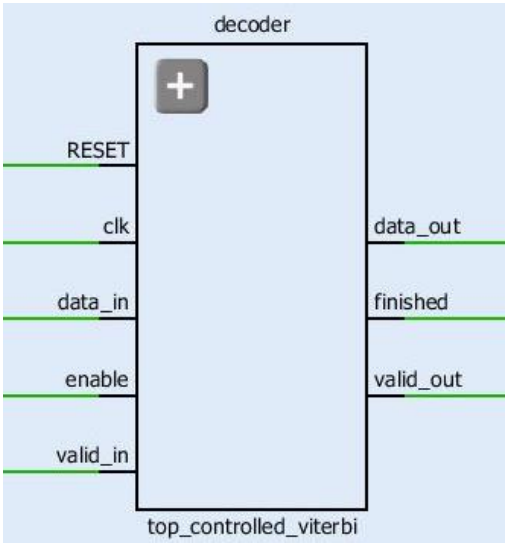


Figure 3-45: Top controlled Decoder

Table 3-13: pins description of Decoder

| Pin | Description |
|---|---|
| data_in | The input data to the block |
| data_out | The output data of the block |
| enable | This signal indicates that the next block (De-Scrambler) is ready to have data |
| finished | This signal indicates that the Decoder is ready to have data |
| valid_in | This signal indicates that current data_in is valid |
| valid_out | This signal indicates that current data_out is valid |

Finally, the LUT utilization of the Viterbi Decoder block is shown in Figure 3-46.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 895 | 0 | 53200 | 1.68 |
| LUT as Logic | 638 | 0 | 53200 | 1.20 |
| LUT as Memory | 257 | 0 | 17400 | 1.48 |
| LUT as Distributed RAM | 257 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 259 | 0 | 106400 | 0.24 |
| Register as Flip Flop | 259 | 0 | 106400 | 0.24 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 8 | 0 | 26600 | 0.03 |
| F8 Muxes | 2 | 0 | 13300 | 0.02 |

Figure 3-46: Decoder LUT utilization

### 3.3.7 De-Scrambler

The block design is the same as that described in section 3.2.1.

The top controlled module of De-Scrambler is as shown in Figure 3-47, and the pins description of the controlled module is shown in Table 3-14.

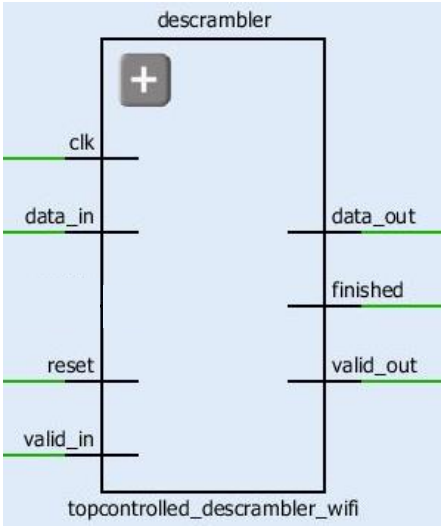


Figure 3-47: Top controlled De-Scrambler

Table 3-14: pins description of De-Scrambler

| Pin | Description |
|---|---|
| data_in | The input data to the block |
| data_out | The output data of the block |
| finished | This signal indicates that the De-Scrambler is ready to have data |
| valid_in | This signal indicates that current data_in is valid |
| valid_out | This signal indicates that current data_out is valid |

Finally, the LUT utilization of the De-Scrambler block is shown in Figure 3-48.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 138 | 0 | 53200 | 0.26 |
| LUT as Logic | 136 | 0 | 53200 | 0.26 |
| LUT as Memory | 2 | 0 | 17400 | 0.01 |
| LUT as Distributed RAM | 2 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 80 | 0 | 106400 | 0.08 |
| Register as Flip Flop | 80 | 0 | 106400 | 0.08 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 3-48: De-Scrambler LUT utilization

The LUT utilization of the WI-FI receiver full chain is shown in Figure 3-49.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 3807 | 0 | 53200 | 7.16 |
| LUT as Logic | 2932 | 0 | 53200 | 5.51 |
| LUT as Memory | 875 | 0 | 17400 | 5.03 |
| LUT as Distributed RAM | 339 | 0 | | |
| LUT as Shift Register | 536 | 0 | | |
| Slice Registers | 2737 | 0 | 106400 | 2.57 |
| Register as Flip Flop | 2737 | 0 | 106400 | 2.57 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 34 | 0 | 26600 | 0.13 |
| F8 Muxes | 2 | 0 | 13300 | 0.02 |

Figure 3-49: WI-FI receiver full chain LUT utilization

The utilization report of the WI-FI full chain is shown in Figure 3-50.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 6380 | 0 | 53200 | 11.99 |
| LUT as Logic | 4907 | 0 | 53200 | 9.22 |
| LUT as Memory | 1473 | 0 | 17400 | 8.47 |
| LUT as Distributed RAM | 407 | 0 | | |
| LUT as Shift Register | 1066 | 0 | | |
| Slice Registers | 4945 | 0 | 106400 | 4.65 |
| Register as Flip Flop | 4945 | 0 | 106400 | 4.65 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 62 | 0 | 26600 | 0.23 |
| F8 Muxes | 2 | 0 | 13300 | 0.02 |

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Block RAM Tile | 7.5 | 0 | 140 | 5.36 |
| RAMB36/FIFO* | 4 | 0 | 140 | 2.86 |
| RAMB36E1 only | 4 | | | |
| RAMB18 | 7 | 0 | 280 | 2.50 |
| RAMB18E1 only | 7 | | | |

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| DSPs | 18 | 0 | 220 | 8.18 |
| DSP48E1 only | 18 | | | |

Figure 3-50: WI-FI full chain utilization report

CHAPTER 4

# LTE Chain

## 4.1 Introduction

LTE is a standard for high-speed wireless communication for mobile phones and data terminals, based on the GSM/EDGE and UMTS/HSPA technologies. It increases the capacity and speed using a different radio interface together with core network improvements. It's a next generation mobile system from the 3GPP with a focus on wireless broadband. LTE is based on OFDM with CP in the downlink, and on SC-FDMA with CP in the uplink. It supports both FDD and TDD duplex modes for transmission on paired and unpaired spectrum.

### 4.1.1 Frame structure

The first frame structure is for FDD and the second one is for TDD (Only FDD is discussed in this thesis). FDD is dividing the frequency into different subcarriers. The distribution of frequencies is as follow:

- 12 sub-carriers per every sub-channel.
- Sub-channel BW is 180KHz
- Subcarrier BW is 15KHz

The distribution for the frame per one subcarrier is shown in Figure 4-1.

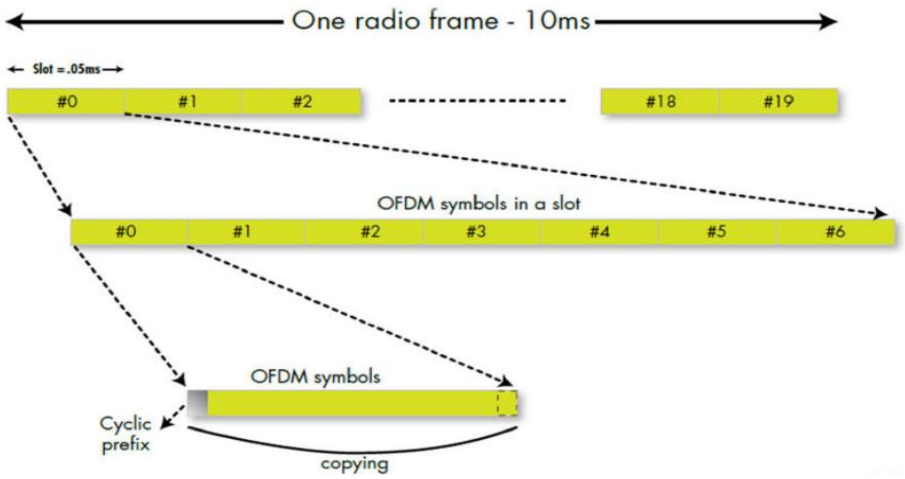


Figure 4-1: LTE Frame structure for FDD

The subcarrier is divided into frames. The time for one frame is 10msec. The frame is divided into sub-frame each with duration 1msec. the sub-frame is divided into two slots. The slot carries the OFDM symbol where the symbol contains both the samples plus the CP. The number of bits per sample depends on the modulation technique used; it may be 16QAM-64QAM ……etc.

As shown in Figure 4-2, the resources of the LTE are divided horizontally to 12 different frequencies represent the subcarriers of LTE. Each frequency is divided into small blocks where the small block represents a symbol and every 7 successive symbols represent a time slot.

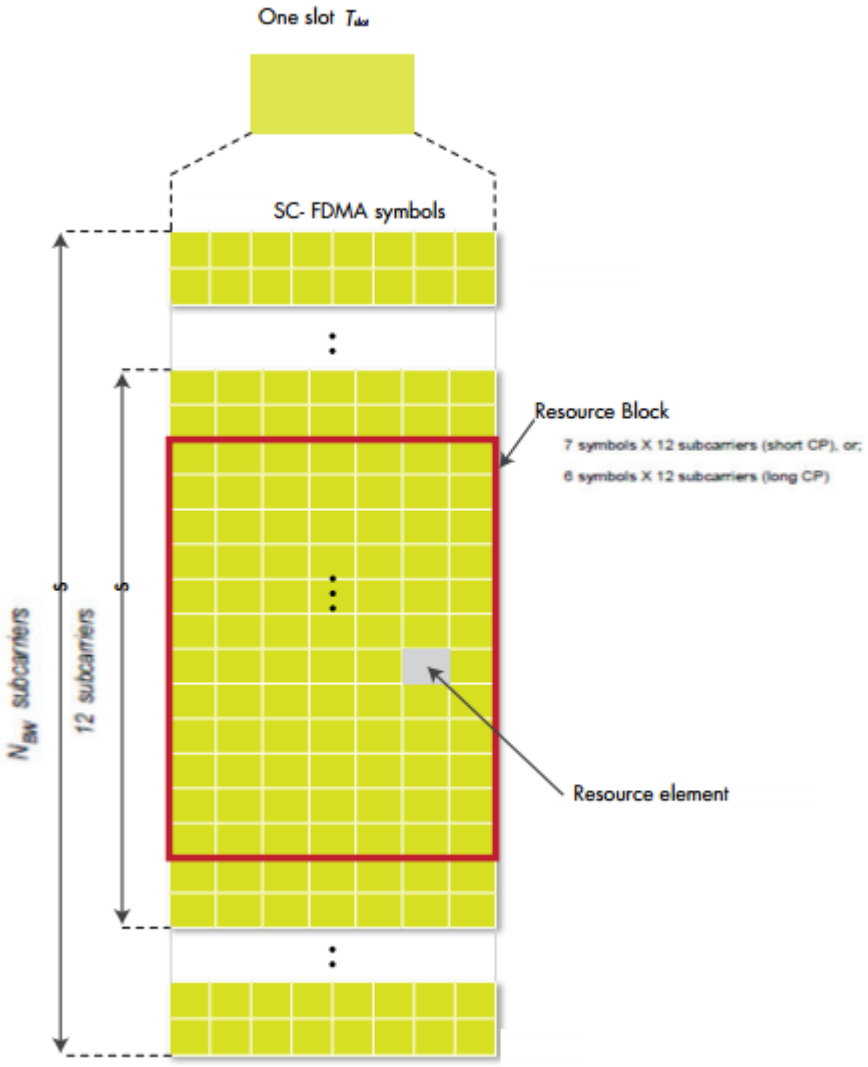


Figure 4-2: Resource elements in LTE

To adjust synchronization and to see the effect of channel on data being transmitted, reference signals are embedded within the data during transmission as

shown in Figure 4-3. Moreover, the reference signal is used during the usage of MIMO antennas concept.

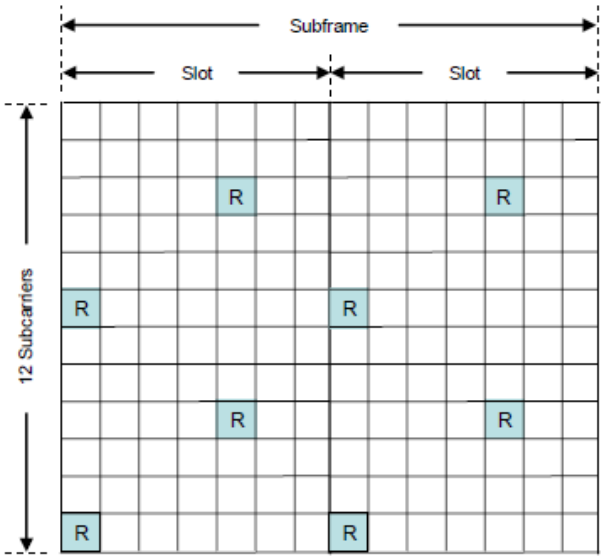


Figure 4-3: Reference signals positions in LTE frame

## 4.2 LTE Transmitter PHY Block Diagram

Since in our project we are working on a kit that holds only one port for antenna. Therefore, the main concept of LTE that consider in MIMO technique is not used. So, the transmitter block diagram can be represented as shown in Figure 4-4.

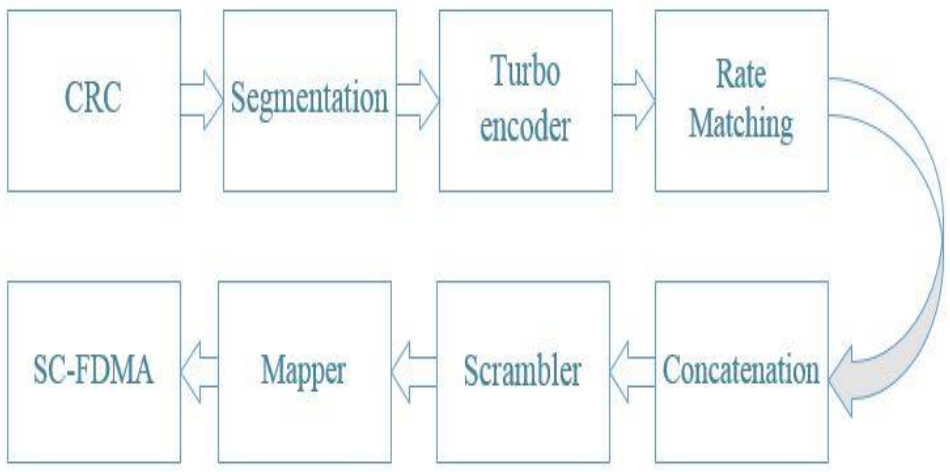


Figure 4-4: LTE Transmitter full chain blocks

In the following sub-sections, each block of the chain is explained in more details _ illustrating its basic idea, showing its interfaces, connections, inputs & outputs & presenting its LUT utilization_.

### 4.2.1 CRC attachment

The CRC block in LTE is the same as in 3G, stated in section 2.2.1 except the polynomial generator which is different than 3G. The polynomial generator equation is

$g_{CRC24A}(D)= D^{24} + D^{23} + D^{18} + D^{17} + D^{14} + D^{11} + D^{10} + D^{7} + D^{6} + D^{5} + D^{4} + D^{3} + D + 1$

or $g_{CRC24B}(D) = D^{24} + D^{23} + D^{6} + D^{5} + D + 1$

or $g_{CRC16}(D) = D^{16} + D^{12} + D^{5} + 1$ or $g_{CRC8}(D) = D^{8} + D^{7} + D^{4} + D^{3} + D + 1$

However, only $g_{CRC24A}(D)$ is implemented. The top controlled module of CRC is as shown in Figure 4-5, A detailed CRC module is as shown in Figure 4-6, and the pins description of the controlled module is shown in Table 4-1.

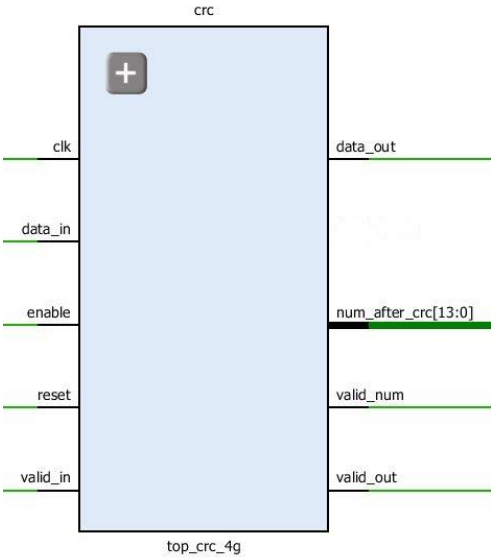


Figure 4-5: Top controlled CRC

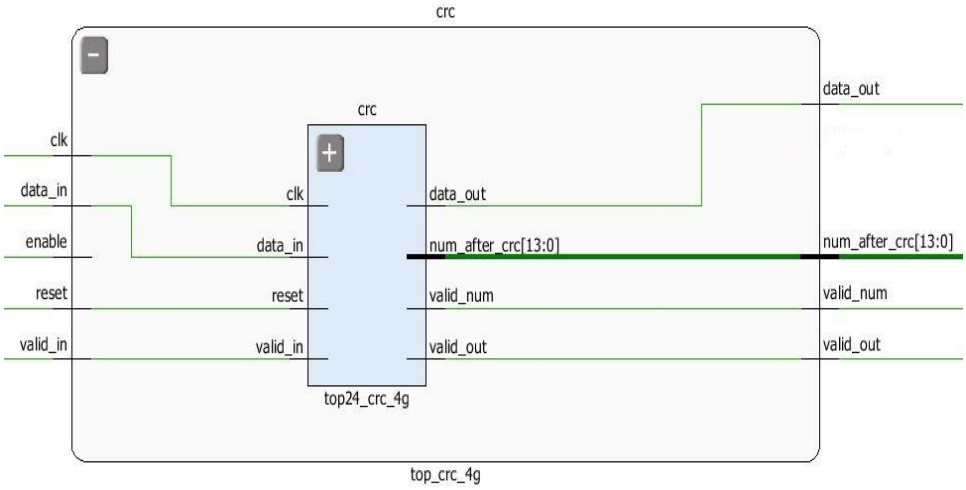


Figure 4-6: Top controlled CRC from inside

Table 4-1: pins description of CRC

| PIN | Description |
|---|---|
| data_in | The input bits to the block |
| data_out | The output data of the block |
| enable | This signal indicates that the next block (Segmentation) is ready to have data |
| valid_num | This signal indicates that the num_after_crc is valid |
| num_after_crc | This signal indicates that total number of bits after CRC |
| valid_in | This signal indicates that the current data_in is valid data |
| valid_out | This signal indicates that the current data_out is valid data |

Finally, the LUT utilization of the CRC block is shown in Figure 4-7.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 89 | 0 | 53200 | 0.17 |
| LUT as Logic | 89 | 0 | 53200 | 0.17 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 75 | 0 | 106400 | 0.07 |
| Register as Flip Flop | 75 | 0 | 106400 | 0.07 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 1 | 0 | 26600 | <0.01 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 4-7: CRC LUT utilization

### 4.2.2 Code block Segmentation

The input bit sequence to the code block segmentation is denoted by:$b_0, b_1, b_2 \ldots\ldots b_{B-1}$. If B is larger than the maximum code block size Z, segmentation of the input bit sequence is performed and an additional CRC sequence of L = 24 bits is attached to each code block. The maximum code block size (Z) is 6144 and the minimum code block size (Z) is 40.

Total number of code blocks C is determined by the following algorithm:

| | | | |
|---|---|---|---|
| if $B <= Z$ | $L = 0$ | Number of code blocks: $C = 1$ | $B' = B$ |
| else | $L = 24$ | $C = \lceil B / (Z - L) \rceil$ | $B' = B + C.L$ |
| end if | | | |

The bits output from code block segmentation, for C! = 0, are denoted by: $C_{r0}$, $C_{r1}$, $C_{r2}$, ……., $C_{r(kr-1)}$ where r is the code block number and $K_r$ is the number of bits for the code block number r.

First segmentation size: $K_+$ = minimum K in Table 2-4 such that C*K > B'

if C = 1 Number of code blocks with length $K_+$ is $C_+ = 1$, $K_- = 0$, $C_- = 0$

else if C > 1 Second segmentation size: $K_-$ = max. K such that $K < K_+$

$\Delta K = K_+ - K_-$

Number of segments of size $K_-$: $C_- = \left\lfloor \frac{C \cdot K_+ - B'}{\Delta K} \right\rfloor$

Number of segments of size $K_+$: $C_+ = C - C_-$

Blocks with size $K_-$ are out first then Blocks with $K_+$

The sequence $C_{r0}$, $C_{r1}$, $C_{r2}$, ……., $C_{r(kr-L-1)}$ is used to calculate the CRC parity bits $p_{r0}$, $p_{r1}$, $p_{r2}$, ……., $p_{r(L-1)}$ with the generator polynomial $g_{CRC24B}(D) = D^{24} + D^{23} + D^6 + D^5 + D + 1$.

end if

Filler bits number added to the first block beginning: $F = C_+ * K_+ + C_- * K_- * B'$

if B < 40 Filler bits are added to the beginning of the code block.

Table 4-2: K Values

| i | K | i | K | i | k |
|---|---|---|---|---|---|
| 1 | 2 | 5 | 32 | 9 | 512 |
| 2 | 4 | 6 | 64 | 10 | 1024 |
| 3 | 8 | 7 | 128 | 11 | 2048 |
| 4 | 16 | 8 | 256 | 12 | 4096 |

This table is simplified version of full k table in standard to simplify the implementation of Segmentation and turbo encoder and also we assumed Z equal 4096 instead of 6144.

For example: if number of input data bits (B) = 8000

The number of blocks (C)= $\lceil 8000 / (4096 - 24) \rceil = 2$

$B' = B + C * L = 8000 + 2*24 = 8048$

$K_+$ = minimum $K$ in Table 2-4 **Error! Reference source not found.** such that K > 4024 = 4096

$K_-$ = maximum K such that K < 4096 = 2048

$\Delta K = 4096 - 2048 = 2048$

Number of segments of size $K_-$: $C_- = \left\lfloor \frac{8192 - 8048}{2048} \right\rfloor = 0$

Number of segments of size $K_+$: $C_+ = 2 - 0 = 2$

$F = 2 * 4096 + 0 * 2048 - 8408 = 144$

| First block (r=0) | | |
|---|---|---|
| 144 filler bits | 3928 bits from $b_0$ $to$ $b_{3927}$ | 24 CRC bits |
| **Second block (r=1)** | | |
| 4072 bits from $b_{3928}$ $to$ $b_{7999}$ | | 24 CRC bits |

Segmentation is implemented as a FSM where Figure 4-8 shows the state diagram of it.

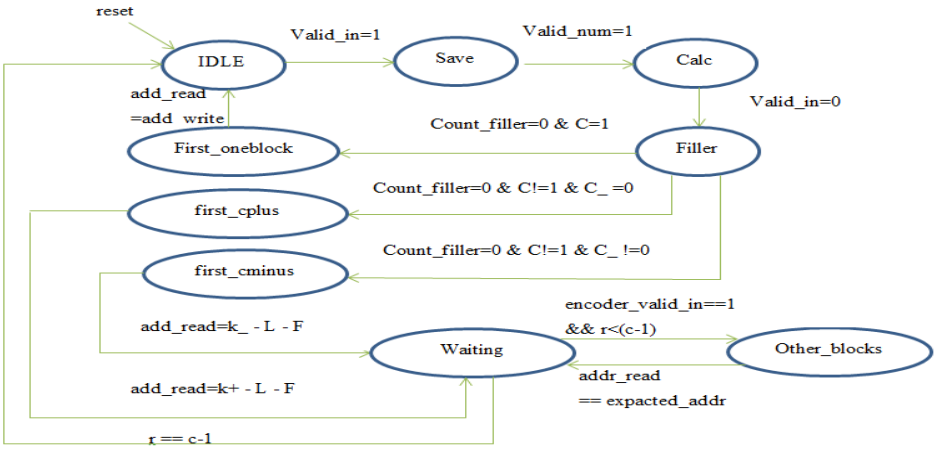


Figure 4-8: Segmentation State Diagram

The top controlled module of Segmentation is as shown in Figure 4-9, A detailed Segmentation module is as shown in Figure 4-10, and the pins description of the controlled module is shown in Table 4-3.

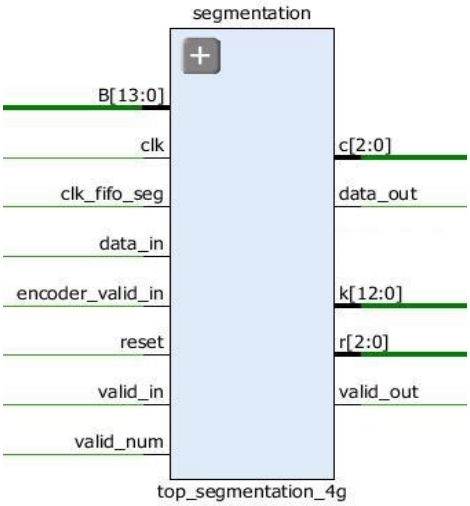


Figure 4-9: Top controlled Segmentation

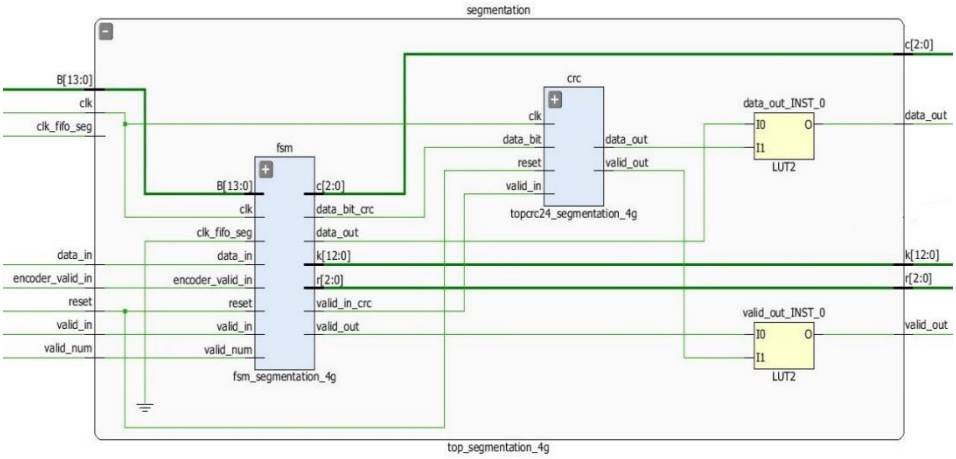


Figure 4-10: Top controlled Segmentation from inside

Table 4-3: pins description of Segmentation

| Pin | Description |
|---|---|
| data_in | Input data to block |
| valid_in | Indicates that input data is valid |
| encoder_valid_in | Enable signal to segmentation block |
| B | Total number of bits coming from CRC block |
| valid_num | Indicates that B is valid |
| data_out | Output data of the indexed block |
| C | Number of blocks |
| R | Index of output code block |
| K | Number of bits per clock |
| valid_out | Indicates that data_out is valid |

Finally, the LUT utilization of the Segmentation block is shown in Figure 4-11.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 538 | 0 | 53200 | 1.01 |
| LUT as Logic | 538 | 0 | 53200 | 1.01 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 220 | 0 | 106400 | 0.21 |
| Register as Flip Flop | 220 | 0 | 106400 | 0.21 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 2 | 0 | 26600 | <0.01 |
| F8 Muxes | 1 | 0 | 13300 | <0.01 |

Figure 4-11: Segmentation LUT utilization

### 4.2.3 Turbo Encoder

The scheme of turbo encoder is a PCCC with two 8-state constituent encoders and one turbo code internal Interleaver. The coding rate of turbo encoder is 1/3. The structure of turbo encoder is illustrated in Figure 4-12. The transfer function of the 8-state constituent code for the PCCC is:

$$G(D) = [1, \frac{g_1(d)}{g_0(d)}] \qquad \text{Where } g_0(D) = 1+D^2+D^3 \ \& \ g_1(D) = 1+D+D^3$$

The initial value of the shift registers of the 8-state constituent encoders shall be all zeros when starting to encode the input bits.

The output from the turbo encoder is $d_k^{(0)} = x_k$ & $d_k^{(1)} = z_k$ & $d_k^{(2)} = z'_k$

For k= 0,1,2, ..., K-1. If the code block to be encoded is the 0-th code block and the number of filler bits is greater than zero, i.e., F > 0, then the encoder shall set $c_k$ = 0, k = 0, ..., (F-1) at its input and shall set $d_k^{(0)}$ = NULL, k = 0, ..., (F-1) and $d_k^{(1)}$ = NULL, k = 0, ..., (F-1) at its output.

The bits input to the turbo encoder are denoted by $c_0$, $c_1$, $c_2$, $c_3$, ..., $c_{K1}$, and the bits output from the first and second 8- state constituent encoders are denoted by $z_0$, $z_1$, $z_2$, $z_3$, …., $z_{k-1}$ and $z_0'$, $z_1'$, $z_2'$, $z_3'$, …., $z_{K-1}'$ respectively. The bits output from the turbo code internal Interleaver are denoted by $c_0'$, $c_1'$, $c_2'$, $c_3'$, …., $c_{k-1}'$ and these bits are to be the input to the second 8-state constituent encoder.

Trellis termination for turbo encoder

Trellis termination is performed by taking the tail bits from the shift register feedback after all information bits are encoded. Tail bits are padded after the encoding of information bits.

The first three tail bits shall be used to terminate the first constituent encoder (upper switch of Figure 4-12 in lower position) while the second constituent encoder is disabled. The last three tail bits shall be used to terminate the second constituent encoder (lower switch of Figure 12-4 in lower position) while the first constituent encoder is disabled.

The transmitted bits for trellis termination shall then be:

$$d_K^{(0)} = x_K,\ d_{K+1}^{(0)} = z_{K+1},\ d_{K+2}^{(0)} = x'_K,\ d_{K+3}^{(0)} = z'_{K+1}$$

$$d_K^{(1)} = z_K,\ d_{K+1}^{(1)} = x_{K+2},\ d_{K+2}^{(1)} = z'_K,\ d_{K+3}^{(1)} = x'_{K+2}$$

$$d_K^{(2)} = x_{K+1},\ d_{K+1}^{(2)} = z_{K+2},\ d_{K+2}^{(2)} = x'_{K+1},\ d_{K+3}^{(2)} = z'_{K+2}$$

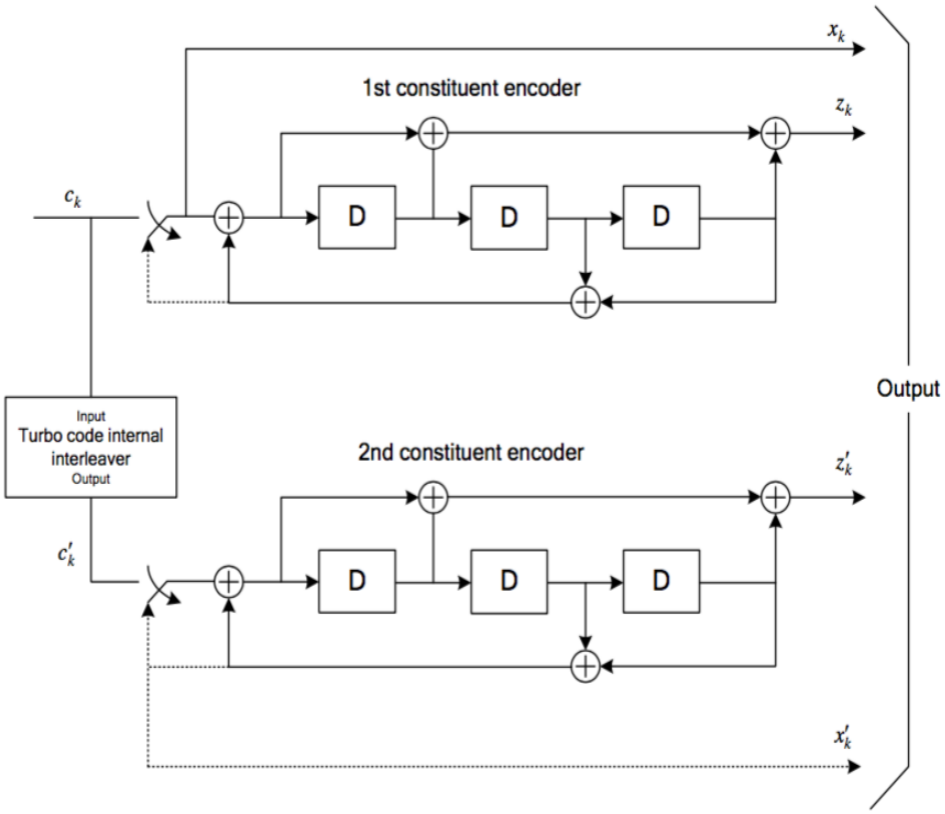


Figure 4-12: Structure of rate 1/3 turbo encoder

Turbo Code Internal Interleaver

The bits input to the turbo code internal Interleaver are denoted by $c_0$, $c_1$, ..., $c_K$ where K is the number of input bits. The bits output from the turbo code internal Interleaver are denoted by $c_0'$, $c_1'$, $c_2'$, $c_3'$, …., $c_{k-1}'$.

The relationship between the input and output bits is as follows:

$$c_i' = c_{\prod(i)},\ i=0, 1, ..., (K-1)$$

where the relationship between the output index i and the input index $\prod(i)$ satisfies the following quadratic form: $\prod(i) = (f_1*i + f_1*i^2) \bmod K$.

The parameters depend on the block size K and are summarized in Figure 4-14. The turbo encoder with even K numbers thus the division would be synthesizable.

The top controlled module of Segmentation is as shown in Figure 4-13, and the pins description of the controlled module is shown in Table 4-4.

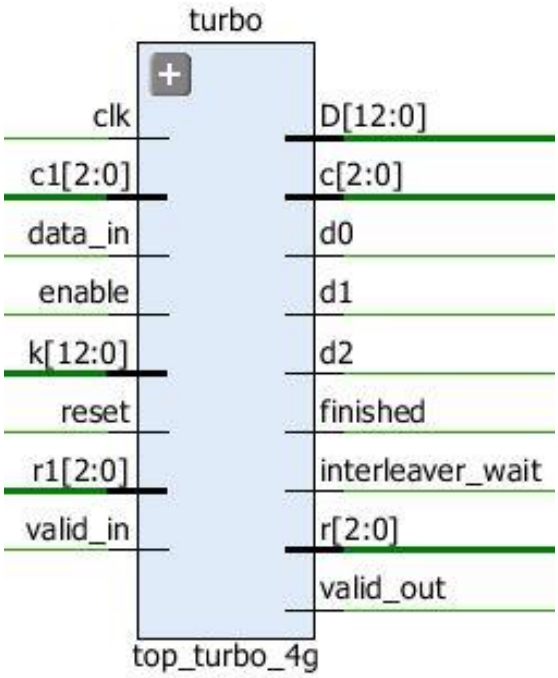


Figure 4-13: Top controlled Turbo Encoder

| *i* | *K* | $f_1$ | $f_2$ | *i* | *K* | $f_1$ | $f_2$ | *i* | *K* | $f_1$ | $f_2$ | *i* | *K* | $f_1$ | $f_2$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 40 | 3 | 10 | 48 | 416 | 25 | 52 | 95 | 1120 | 67 | 140 | 142 | 3200 | 111 | 240 |
| 2 | 48 | 7 | 12 | 49 | 424 | 51 | 106 | 96 | 1152 | 35 | 72 | 143 | 3264 | 443 | 204 |
| 3 | 56 | 19 | 42 | 50 | 432 | 47 | 72 | 97 | 1184 | 19 | 74 | 144 | 3328 | 51 | 104 |
| 4 | 64 | 7 | 16 | 51 | 440 | 91 | 110 | 98 | 1216 | 39 | 76 | 145 | 3392 | 51 | 212 |
| 5 | 72 | 7 | 18 | 52 | 448 | 29 | 168 | 99 | 1248 | 19 | 78 | 146 | 3456 | 451 | 192 |
| 6 | 80 | 11 | 20 | 53 | 456 | 29 | 114 | 100 | 1280 | 199 | 240 | 147 | 3520 | 257 | 220 |
| 7 | 88 | 5 | 22 | 54 | 464 | 247 | 58 | 101 | 1312 | 21 | 82 | 148 | 3584 | 57 | 336 |
| 8 | 96 | 11 | 24 | 55 | 472 | 29 | 118 | 102 | 1344 | 211 | 252 | 149 | 3648 | 313 | 228 |
| 9 | 104 | 7 | 26 | 56 | 480 | 89 | 180 | 103 | 1376 | 21 | 86 | 150 | 3712 | 271 | 232 |
| 10 | 112 | 41 | 84 | 57 | 488 | 91 | 122 | 104 | 1408 | 43 | 88 | 151 | 3776 | 179 | 236 |
| 11 | 120 | 103 | 90 | 58 | 496 | 157 | 62 | 105 | 1440 | 149 | 60 | 152 | 3840 | 331 | 120 |
| 12 | 128 | 15 | 32 | 59 | 504 | 55 | 84 | 106 | 1472 | 45 | 92 | 153 | 3904 | 363 | 244 |
| 13 | 136 | 9 | 34 | 60 | 512 | 31 | 64 | 107 | 1504 | 49 | 846 | 154 | 3968 | 375 | 248 |
| 14 | 144 | 17 | 108 | 61 | 528 | 17 | 66 | 108 | 1536 | 71 | 48 | 155 | 4032 | 127 | 168 |
| 15 | 152 | 9 | 38 | 62 | 544 | 35 | 68 | 109 | 1568 | 13 | 28 | 156 | 4096 | 31 | 64 |
| 16 | 160 | 21 | 120 | 63 | 560 | 227 | 420 | 110 | 1600 | 17 | 80 | 157 | 4160 | 33 | 130 |
| 17 | 168 | 101 | 84 | 64 | 576 | 65 | 96 | 111 | 1632 | 25 | 102 | 158 | 4224 | 43 | 264 |
| 18 | 176 | 21 | 44 | 65 | 592 | 19 | 74 | 112 | 1664 | 183 | 104 | 159 | 4288 | 33 | 134 |
| 19 | 184 | 57 | 46 | 66 | 608 | 37 | 76 | 113 | 1696 | 55 | 954 | 160 | 4352 | 477 | 408 |
| 20 | 192 | 23 | 48 | 67 | 624 | 41 | 234 | 114 | 1728 | 127 | 96 | 161 | 4416 | 35 | 138 |
| 21 | 200 | 13 | 50 | 68 | 640 | 39 | 80 | 115 | 1760 | 27 | 110 | 162 | 4480 | 233 | 280 |
| 22 | 208 | 27 | 52 | 69 | 656 | 185 | 82 | 116 | 1792 | 29 | 112 | 163 | 4544 | 357 | 142 |
| 23 | 216 | 11 | 36 | 70 | 672 | 43 | 252 | 117 | 1824 | 29 | 114 | 164 | 4608 | 337 | 480 |
| 24 | 224 | 27 | 56 | 71 | 688 | 21 | 86 | 118 | 1856 | 57 | 116 | 165 | 4672 | 37 | 146 |
| 25 | 232 | 85 | 58 | 72 | 704 | 155 | 44 | 119 | 1888 | 45 | 354 | 166 | 4736 | 71 | 444 |
| 26 | 240 | 29 | 60 | 73 | 720 | 79 | 120 | 120 | 1920 | 31 | 120 | 167 | 4800 | 71 | 120 |
| 27 | 248 | 33 | 62 | 74 | 736 | 139 | 92 | 121 | 1952 | 59 | 610 | 168 | 4864 | 37 | 152 |
| 28 | 256 | 15 | 32 | 75 | 752 | 23 | 94 | 122 | 1984 | 185 | 124 | 169 | 4928 | 39 | 462 |
| 29 | 264 | 17 | 198 | 76 | 768 | 217 | 48 | 123 | 2016 | 113 | 420 | 170 | 4992 | 127 | 234 |
| 30 | 272 | 33 | 68 | 77 | 784 | 25 | 98 | 124 | 2048 | 31 | 64 | 171 | 5056 | 39 | 158 |
| 31 | 280 | 103 | 210 | 78 | 800 | 17 | 80 | 125 | 2112 | 17 | 66 | 172 | 5120 | 39 | 80 |
| 32 | 288 | 19 | 36 | 79 | 816 | 127 | 102 | 126 | 2176 | 171 | 136 | 173 | 5184 | 31 | 96 |
| 33 | 296 | 19 | 74 | 80 | 832 | 25 | 52 | 127 | 2240 | 209 | 420 | 174 | 5248 | 113 | 902 |
| 34 | 304 | 37 | 76 | 81 | 848 | 239 | 106 | 128 | 2304 | 253 | 216 | 175 | 5312 | 41 | 166 |
| 35 | 312 | 19 | 78 | 82 | 864 | 17 | 48 | 129 | 2368 | 367 | 444 | 176 | 5376 | 251 | 336 |
| 36 | 320 | 21 | 120 | 83 | 880 | 137 | 110 | 130 | 2432 | 265 | 456 | 177 | 5440 | 43 | 170 |
| 37 | 328 | 21 | 82 | 84 | 896 | 215 | 112 | 131 | 2496 | 181 | 468 | 178 | 5504 | 21 | 86 |
| 38 | 336 | 115 | 84 | 85 | 912 | 29 | 114 | 132 | 2560 | 39 | 80 | 179 | 5568 | 43 | 174 |
| 39 | 344 | 193 | 86 | 86 | 928 | 15 | 58 | 133 | 2624 | 27 | 164 | 180 | 5632 | 45 | 176 |
| 40 | 352 | 21 | 44 | 87 | 944 | 147 | 118 | 134 | 2688 | 127 | 504 | 181 | 5696 | 45 | 178 |
| 41 | 360 | 133 | 90 | 88 | 960 | 29 | 60 | 135 | 2752 | 143 | 172 | 182 | 5760 | 161 | 120 |
| 42 | 368 | 81 | 46 | 89 | 976 | 59 | 122 | 136 | 2816 | 43 | 88 | 183 | 5824 | 89 | 182 |
| 43 | 376 | 45 | 94 | 90 | 992 | 65 | 124 | 137 | 2880 | 29 | 300 | 184 | 5888 | 323 | 184 |
| 44 | 384 | 23 | 48 | 91 | 1008 | 55 | 84 | 138 | 2944 | 45 | 92 | 185 | 5952 | 47 | 186 |
| 45 | 392 | 243 | 98 | 92 | 1024 | 31 | 64 | 139 | 3008 | 157 | 188 | 186 | 6016 | 23 | 94 |
| 46 | 400 | 151 | 40 | 93 | 1056 | 17 | 66 | 140 | 3072 | 47 | 96 | 187 | 6080 | 47 | 190 |
| 47 | 408 | 155 | 102 | 94 | 1088 | 171 | 204 | 141 | 3136 | 13 | 28 | 188 | 6144 | 263 | 480 |

Figure 4-14: Turbo code internal Interleaver parameters

Table 4-4: pins description of Turbo Encoder

| Pin | Description |
|---|---|
| c | This signal indicates the number of code blocks |
| c1 | This signal indicates the number of code blocks from the segmentation |
| D | This signal indicates the block size after turbo encoder |
| d0 | The output data of the block [Turbo output (0)] |
| d1 | The output data of the block [Turbo output (1)] |
| d2 | The output data of the block [Turbo output (2)] |
| data_in | The input data to the block |
| enable | This signal indicates that the next block (Rate matching) is ready to have data |
| Finished | This signal indicates that the Encoder is ready to have data |
| k | This signal indicates the block size of data |
| r | This signal indicates the block index as output |
| r1 | This signal indicates the block index as input |
| valid_in | This signal indicates that current data_in is valid |
| valid_out | This signal indicates that current data_out is valid |

Finally, the LUT utilization of the Turbo Encoder block is shown in Figure 4-15.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 436 | 0 | 53200 | 0.82 |
| LUT as Logic | 436 | 0 | 53200 | 0.82 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 211 | 0 | 106400 | 0.20 |
| Register as Flip Flop | 173 | 0 | 106400 | 0.16 |
| Register as Latch | 38 | 0 | 106400 | 0.04 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 4-15: Turbo Encoder LUT utilization

### 4.2.4 Rate matching for turbo coded transport channels

The rate matching for turbo coded transport channels is defined per coded block and consists of interleaving the three information bit streams $d_k^{(0)}$, $d_k^{(1)}$ and $d_k^{(2)}$, followed by the collection of bits and the generation of a circular buffer as depicted in Figure 16-4.

The bit stream $d_k^{(0)}$ is interleaved according to the sub-block Interleaver defined in section 4.2.4.1 with an output sequence defined as $v_0^{(0)}, v_1^{(0)}, v_2^{(0)}, \ldots\ldots v_{K_\pi-1}^{(0)}$.

The bit stream $d_k^{(1)}$ is interleaved according to the sub-block Interleaver defined in section 4.2.4.1 with an output sequence defined as $v_0^{(1)}, v_1^{(1)}, v_2^{(1)}, \ldots\ldots v_{K_\pi-1}^{(1)}$.

The bit stream $d_k^{(2)}$ is interleaved according to the sub-block Interleaver defined in section 4.2.4.1 with an output sequence defined as $v_0^{(2)}, v_1^{(2)}, v_2^{(2)}, \ldots\ldots v_{K_\pi-1}^{(2)}$.

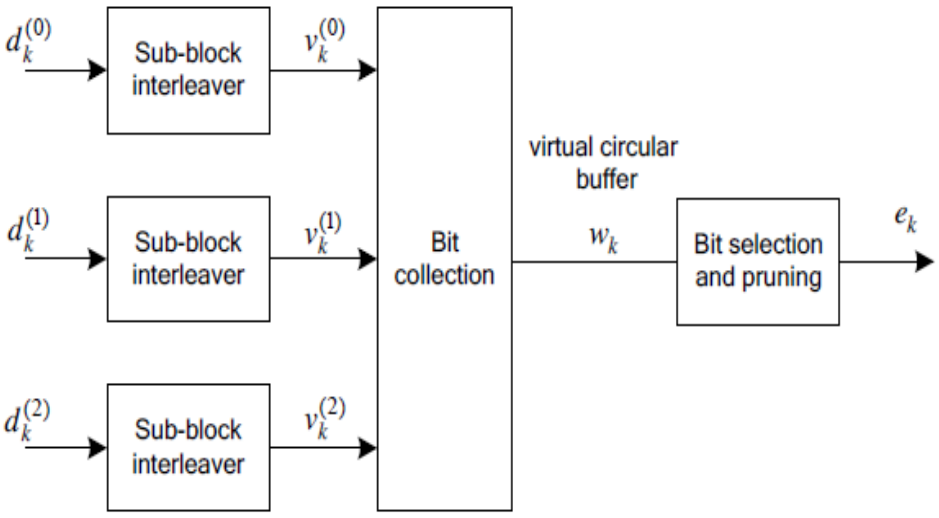


Figure 4-16: Rate matching for turbo coded transport channels

### 4.2.4.1 Sub-block Interleaver

The bits input to the block Interleaver are denoted by $d_0^{(i)}, d_1^{(i)}, d_2^{(i)}, \ldots\ldots, d_{D-1}^{(i)}$,where D is the number of bits. The output bit sequence from the block interleaver is derived as follows:

1. Assign $C_{subblock}^{TC} = 32$ to be the number of columns of the matrix. The columns of the matrix are numbered 0,1,2, $\ldots\ldots, C_{subblock}^{TC} - 1$ from left to right.
2. Determine the number of rows of the matrix $R_{subblock}^{TC}$, by finding minimum integer $R_{subblock}^{TC}$ such that:

$$D \leq R_{subblock}^{TC} * C_{subblock}^{TC}$$

The rows of rectangular matrix are numbered 0,1,2, $\ldots\ldots, R_{subblock}^{TC} - 1$ from top to bottom.

3. $R_{subblock}^{TC} * C_{subblock}^{TC} > D$, then $N_D = R_{subblock}^{TC} * C_{subblock}^{TC} - D$ dummy bits are padded such that $y_k$ = <NULL> for $k = 0,1,2,\ldots..,N_D - 1$. Then, $y_{N_{D+k}} = d_k^{(i)}$ , k = 0, 1, …, D-1, and the bit sequence $y_k$ is written into the $R_{subblock}^{TC} * C_{subblock}^{TC}$ matrix row by row starting with bit y0 in column 0 of row 0:

$$\begin{bmatrix} y_0 & y_1 & y_2 & \cdots & y_{C_{subblock}^{TC}-1} \\ y_{C_{subblock}^{TC}} & y_{C_{subblock}^{TC}+1} & y_{C_{subblock}^{TC}+2} & \cdots & y_{2C_{subblock}^{TC}-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{(R_{subblock}^{TC}-1)\times C_{subblock}^{TC}} & y_{(R_{subblock}^{TC}-1)\times C_{subblock}^{TC}+1} & y_{(R_{subblock}^{TC}-1)\times C_{subblock}^{TC}+2} & \cdots & y_{(R_{subblock}^{TC}\times C_{subblock}^{TC}-1)} \end{bmatrix}$$

4. For $d_k^{(0)}$ and $d_k^{(1)}$: Perform the inter-column permutation for the matrix based on the pattern P(j) that is shown in Table 5-4, where P(j) is the original column position of the j[th] permuted column. After permutation of the columns, the inter-column permuted ($R_{subblock}^{TC} * C_{subblock}^{TC}$) matrix is equal to

$$\begin{bmatrix} y_{P(0)} & y_{P(1)} & y_{P(2)} & \cdots & y_{P(C_{subblock}^{TC}-1)} \\ y_{P(0)+C_{subblock}^{TC}} & y_{P(1)+C_{subblock}^{TC}} & y_{P(2)+C_{subblock}^{TC}} & \cdots & y_{P(C_{subblock}^{TC}-1)+C_{subblock}^{TC}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{P(0)+(R_{subblock}^{TC}-1)\times C_{subblock}^{TC}} & y_{P(1)+(R_{subblock}^{TC}-1)\times C_{subblock}^{TC}} & y_{P(2)+(R_{subblock}^{TC}-1)\times C_{subblock}^{TC}} & \cdots & y_{P(C_{subblock}^{TC}-1)+(R_{subblock}^{TC}-1)\times C_{subblock}^{TC}} \end{bmatrix}$$

5. The output of the block Interleaver is the bit sequence read out column by column from the inter-column permuted ($R_{subblock}^{TC} * C_{subblock}^{TC}$) matrix. The bits after sub-block interleaving are denoted by $v_0^{(i)}, v_1^{(i)}, v_2^{(i)}, \ldots\ldots\ldots v_{K_\pi-1}^{(i)}$ where $v_0^{(i)}$ corresponds to $y_{p(0)}$, $v_1^{(i)}$ corresponds to $y_{p(0)+C_{subblock}^{TC}}$ ... and $k_\pi = (R_{subblock}^{TC} * C_{subblock}^{TC})$.

For $d_k^{(2)}$:

1. The output of the sub-block Interleaver is denoted by $v_0^{(2)}, v_1^{(2)}, v_2^{(2)}, \ldots\ldots\ldots v_{K_\pi-1}^{(2)}$ , where $v_k^{(2)} = y_{\pi(k)}$ and where

$$\pi(k) = \left( P\left( Floor\left( \frac{k}{R_{subblock}^{TC}} \right) \right) + C_{subblock}^{TC} * (k \bmod R_{subblock}^{TC}) + 1 \right) mod\ k_\pi$$

The permutation function P is defined in Table 4-5.

Table 4-5: Inter-column permutation pattern for sub-block Interleaver

| Number of columns $C_{subblock}^{TC}$ | Inter-column permutation pattern $< P(0), P(1),\ldots,P(C_{subblock}^{TC}-1) >$ |
|---|---|
| 32 | < 0, 16, 8, 24, 4, 20, 12, 28, 2, 18, 10, 26, 6, 22, 14, 30, 1, 17, 9, 25, 5, 21, 13, 29, 3, 19, 11, 27, 7, 23, 15, 31 > |

#### 4.2.4.2 Bit Collection

The input bits to the bit collection block is the output bits from the three sub-block Interleaver and the block output can be represented by virtual circular buffer as shown in Figure 4-16.

The circular buffer of length $K_w = 3K_\pi$ for the r[th] coded block is generated as follows:

$$W_k = v_k^{(0)} \qquad \text{For } k = 0,1, \dots \dots K_\pi - 1$$

$$W_{K_\pi + 2k} = v_k^{(1)} \qquad \text{For } k = 0,1, \dots \dots K_\pi - 1$$

$$W_{K_\pi + 2k + 1} = v_k^{(2)} \qquad \text{For } k = 0,1, \dots \dots K_\pi - 1$$

#### 4.2.4.3 Bit selection

It is the last block in the block diagram of the rate matching, the block has its input from the output of the bit collection as shown in Figure 4-16 and it is used to remove the dummy bits from the bit collection output according to the following calculations:

Denote the soft buffer size for the r-th code block by $N_{cb}$ bits. For UL-SCH, MCH, SL-SCH and SL-DCH transport channels $N_{cb} = K_w$.

Define by G the total number of bits available for the transmission of one transport block.

Denoting by E the rate matching output sequence length for the r[th] coded block, and $rv_{idx}$ the redundancy version number for this transmission ($rv_{idx}$ = 0), the rate matching output bit sequence is $e_k$ , k = 0, 1, ..., E-1.

Set $G' = G/(N_L . Q_m)$ where $Q_m$ is equal to 2 for QPSK, 4 for 16QAM, 6 for 64QAM and 8 for 256QAM and where $N_L$= 2 for transmit diversity.

Set γ =$G' \ mod \ C$ , where C is the number of code blocks from segmentation section.

If $r \leq C - \gamma - 1$ Set $E = N_L . Q_m . Floor(G'/C)$

else Set $E = N_L . Q_m . ceil(G'/C)$

end if

Set $k_0 = R_{subblock}^{TC} . \left(2 . ceil\left(\frac{N_{cb}}{8R_{subblock}^{TC}}\right) . rv_{idx} + 2\right)$

Set $k = 0 \; and \; j = 0$

while k < E

If $w_{(k_0+j) mod \, N_{cb}} \neq < NULL >$

$e_k = w_{(k_0+j) mod \, N_{cb}}$

$k = k + 1$

endif

$j = j + 1$

end while

The top controlled module of Rate matching is as shown in Figure 4-17, and the pins description of the controlled module is shown in Table 4-6.

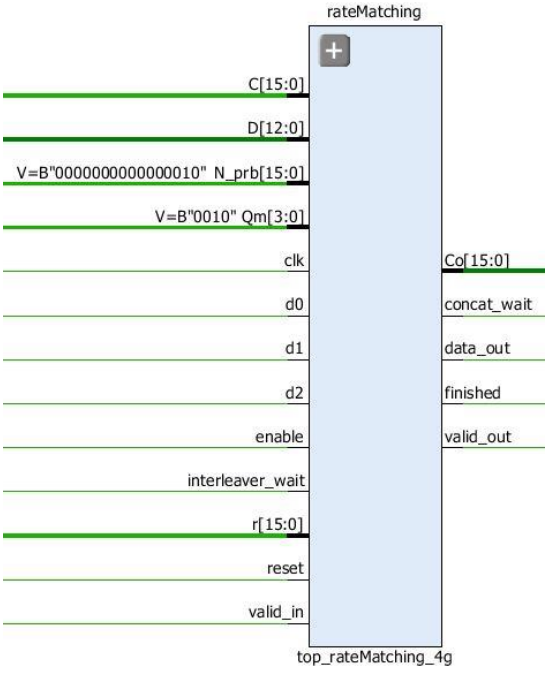


Figure 4-17: Top controlled Rate matching

Table 4-6: pins description of Rate matching

| Pin | Description |
|---|---|
| do,d1,d2 | Input data from encoder |
| valid_in | Indicates that input data is valid |
| enable | Enable signal |
| D | Total number in code block after encoding |
| data_out | Output data of the indexed block |
| C,Co | Number of code blocks |
| r | Index of input code block |
| interleaver_wait | Indicates that data_in should be neglected as encoder is busy |
| valid_out | Indicates that output data is valid |
| concat_wait | Indicates that data_out should be neglected as rate_matching is busy |
| finished | Block finished and can receive new data |
| Qm | Indicates number of bits per symbol (Ex.: Qm = 2 if QPSK is used) |
| N_prb | Number of resource blocks assigned to user |

Finally, the LUT utilization of the Rate matching block is shown in Figure 4-18.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 3627 | 0 | 53200 | 6.82 |
| LUT as Logic | 3627 | 0 | 53200 | 6.82 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 587 | 0 | 106400 | 0.55 |
| Register as Flip Flop | 555 | 0 | 106400 | 0.52 |
| Register as Latch | 32 | 0 | 106400 | 0.03 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 4-18: Rate matching LUT utilization

### 4.2.5 Code block concatenation

The block design is the same as that described in section 2.2.4.

The top controlled module of concatenation is as shown in Figure 4-19, and the pins description of the controlled module is shown in Table 4-7.

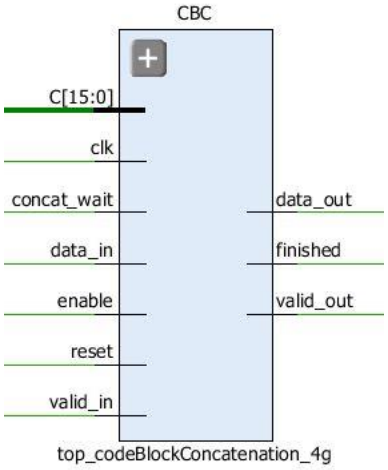


Figure 4-19: Top controlled Concatenation

Table 4-7: pins description of Concatenation

| PIN | Description |
|---|---|
| c | This signal indicates the number of code blocks from the segmentation block |
| data_in | The input bits to the block |
| data_out | The output bits of the block |
| enable | This signal indicates that the next block ( Scrambler ) is ready to have data |
| finished | This signal indicates that the Concatenation block is ready to have a new frame |
| valid_in | This signal indicates that current data_in is valid data |
| valid_out | This signal indicates that current data_out is valid data |

Finally, the LUT utilization of the Concatenation block is shown in Figure 4-20.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 110 | 0 | 53200 | 0.21 |
| LUT as Logic | 110 | 0 | 53200 | 0.21 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 73 | 0 | 106400 | 0.07 |
| Register as Flip Flop | 73 | 0 | 106400 | 0.07 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 4-20: Concatenation LUT utilization

### 4.2.6 Scrambler

Scrambler is used to randomize the bits, prevent long sequences of 1s or 0s to keep synchronization.

For each code-word q, the block of bits $b^{(q)}(0)$, ..., $b^{(q)}(M_{bit}^{(q)}-1)$, where $M_{bit}^{(q)}$ is the number of bits transmitted in code-word q on the physical uplink shared channel in one sub-frame, shall be scrambled with a UE-specific scrambling sequence prior to modulation, resulting in a block of scrambled bits $b^{\sim(q)}(0)$, ..., $b^{\sim(q)}(M_{bit}^{(q)}-1)$ according to the following pseudo code:

```
Set i = 0
While i < M_bit^(q)
  if b^(q)(i) = x     // ACK/NACK or Rank Indication placeholder bits
        b~^(q)(i) = 1
  else
        if b^(q)(i) = y     // ACK/NACK or Rank Indication repetition placeholder
                            // bits
              b~^(q)(i) = b~^(q)(i-1)
        else     // Data or channel quality coded bits, Rank Indication coded bits
                 // or ACK/NACK coded bits
              b~^(q)(i) = (b^(q)(i) + c^(q)(i)) mod 2
        end if
  end if
  i = i + 1
  end while
```

For simplicity we assume that $b^{\sim(q)}(i) = (b^{(q)}(i) + c^{(q)}(i)) \bmod 2$.

The Pseudo-random sequences here are defined by a length-31 Gold sequence. The output sequence c(n) of length $M_{PN}$, where n = 0, 1, ..., $M_{PN}$ -1, is defined by

$$c(n) = (x_1(n + N_c) + x_2(n + N_c)) \bmod 2$$

$$x_1(n + 31) = (x_1(n + 3) + x_1(n)) \bmod 2$$

$$x_2(n + 31) = (x_2(n + 3) + x_2(n + 2) + x_2(n + 1) + x_2(n)) \bmod 2$$

where $N_c = 1600$ and the first m-sequence shall be initialized with $x_1(0) = 1$, $x_1(n) = 0$, n = 1, 2, ..., 30. The initialization of the second m-sequence is denoted by $c_{init}$ where $c_{init} = \sum_{i=0}^{30} x_2(i) * 2^i$ with the value depending on the application of the sequence. Finally, the scrambling sequence generator shall be initialized with

$$C_{init} = n_{RNTI} \cdot 2^{14} + q \cdot 2^{13} + \text{floor}(n_s / 2) \cdot 2^9 + N_{ID}$$

At the start of each sub-frame where $n_{RNTI}$ corresponds to the RNTI associated with the PUSCH transmission, $n_s$ is the index of the sub-frame & $N_{ID}$ is the cell ID.

The top controlled module of Scrambler is as shown in Figure 4-21, A detailed Scrambler module is as shown in Figure 4-22, and the pins description of the controlled module is shown in Table 4-8.

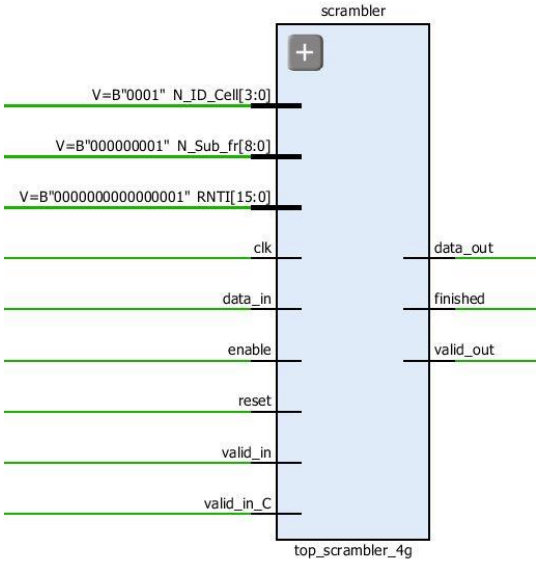


Figure 4-21: Top controlled Scrambler

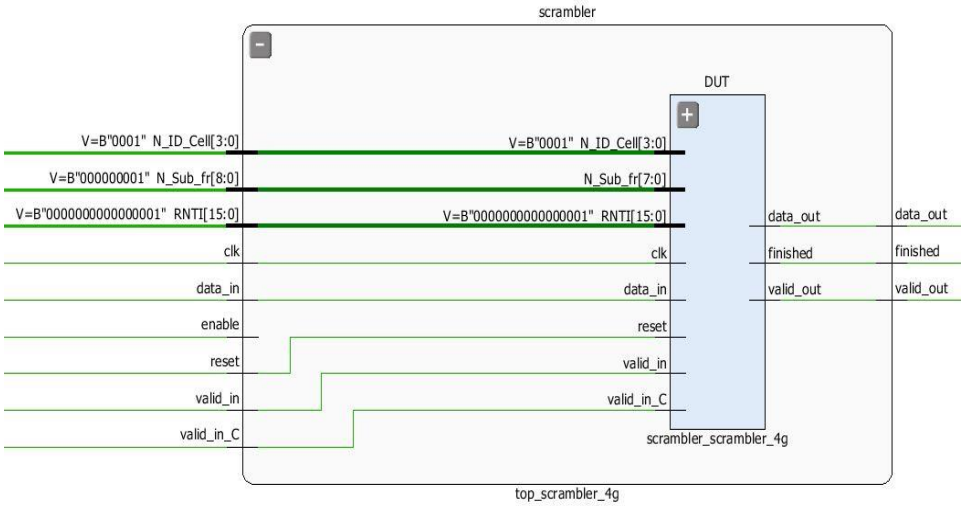


Figure 4-22: Top controlled Scrambler from inside

Table 4-8: pins description of Scrambler

| Pin | Description |
|---|---|
| data_in | Input data from rate matching |
| valid_in | Indicates that input data is valid |
| N_ID_Cell, RNTI,N_Sub_fr | MAC layer parameters |
| data_out | Output data of the indexed block |
| valid_in_c | Indicates that MAC layer parameters are valid |
| valid_out | Indicates that output data is valid |
| Finished | Block finished and can receive new data |
| N_prb | Number of resource blocks assigned to user |
| Enable | Enable signal |

Finally, the LUT utilization of the Scrambler block is shown in Figure 4-23.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 441 | 0 | 53200 | 0.83 |
| LUT as Logic | 441 | 0 | 53200 | 0.83 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 222 | 0 | 106400 | 0.21 |
| Register as Flip Flop | 222 | 0 | 106400 | 0.21 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 4-23: Scrambler LUT utilization

### 4.2.7 Mapper

The block design is the same as that described in section 2.2.7. the only differences exist in that the used modulation scheme is QPSK not BPSK.

The top controlled module of Mapper is as shown in Figure 4-24, A detailed Mapper module is as shown in Figure 4-25, and the pins description of the controlled module is shown in Table 4-9.

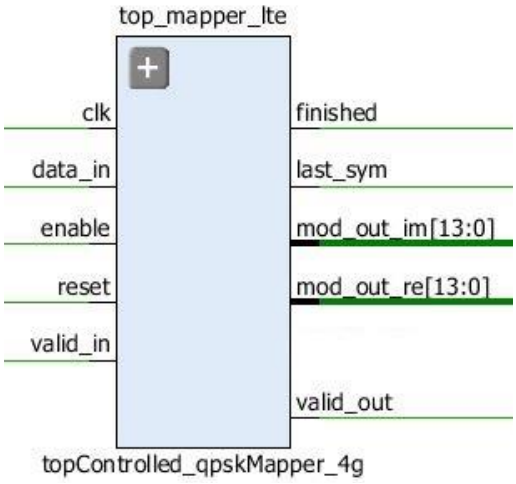


Figure 4-24: Top controlled Mapper

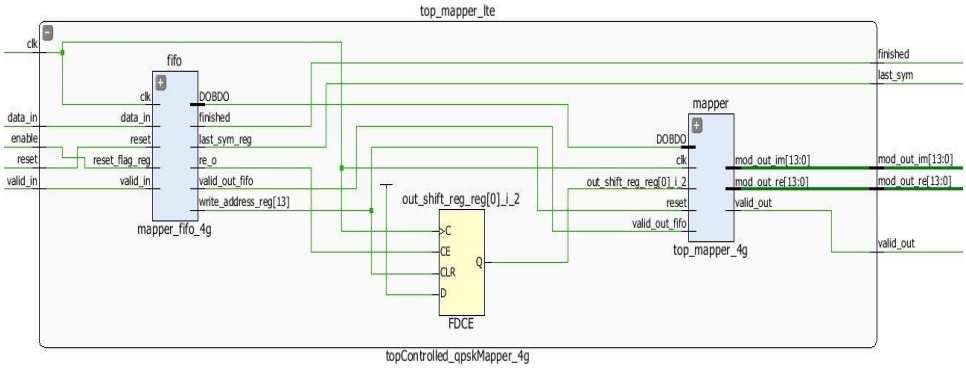


Figure 4-25: Top controlled Mapper from inside

Table 4-9: pins description of Mapper

| Pin | Description |
|---|---|
| data_in | The input data to the block |
| enable | This signal indicates that the next block (SC-FDMA) is ready to have data |
| finished | This signal indicates that the Mapper is ready to have data |
| last_sym | This signal indicates that these patch of symbols are the last to be processed |
| mod_out_im | The imaginary data output of the block |
| mod_out_re | The real data output of the block |
| valid_in | This signal indicates that the input data is valid data |
| valid_out | This signal indicates that the output data is valid data |

Finally, the LUT utilization of the Mapper block is shown in Figure 4-26.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 80 | 0 | 53200 | 0.15 |
| LUT as Logic | 80 | 0 | 53200 | 0.15 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 69 | 0 | 106400 | 0.06 |
| Register as Flip Flop | 69 | 0 | 106400 | 0.06 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 4-26: Mapper LUT utilization

### 4.2.8 SC-FDMA

LTE uplink uses SC-FDMA which is a modified form of the OFDM with similar throughput performance and complexity, SC-FDMA is viewed as DFT-coded OFDM where time-domain symbols are transformed to frequency domain symbols and then go through the standard OFDM modulation as shown in Figure 4-27.

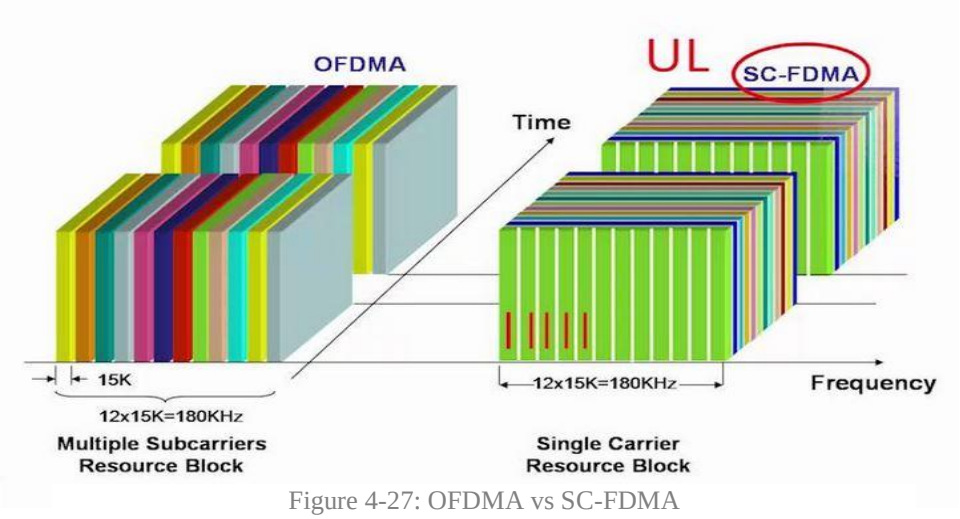


Figure 4-27: OFDMA vs SC-FDMA

SC-FDMA has all the advantages of OFDM like robustness against multi-path signal propagation, the block diagram for the SC-FDMA is shown in Figure 4-28.


Figure 4-28: SC-FDMA block diagram

The IFFT subcarriers are grouped into sets of 12 sub-carriers, each group is called a resource block. The main advantage of SC-FDMA is the low PAPR of the transmitted signal, PAPR is a big concern for user equipment, as PAPR relates to the power amplifier efficiency as low PAPR allows the power amplifier to operate close to the saturation region resulting in high efficiency that is why SC-FDMA is the preferred technology for user terminals.

we chose the 128-point IFFT to facilitate our testing process, we also chose the extended cyclic prefix. So, for our 128 point IFFT we have 32 cyclic prefix and our sampling frequency is 1.92 MHz.

The top controlled module of SC-FDMA is as shown in Figure 4-29, A detailed SC-FDMA module is as shown in Figure 4-30, and the pins description of the controlled module is the same as that in section 3.2.6 where the different pins are start_rb and num_of_rbs, start_rb defines the starting resource block that will be used, and num_of_rbs defines the number of resource blocks to use and the IFFT_clk is 1.92 MHz.

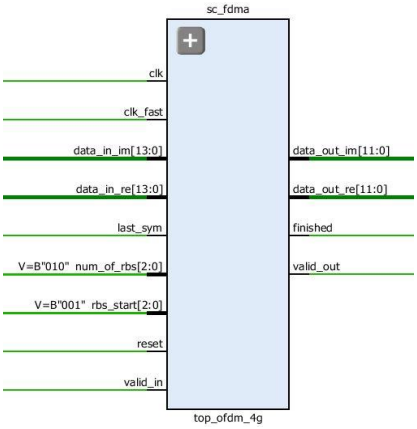


Figure 4-29: Top controlled SC-FDMA

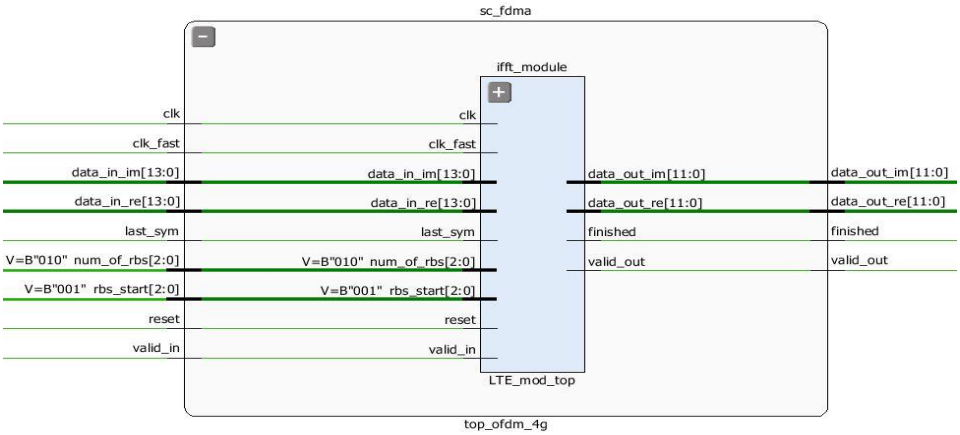


Figure 4-30: Top controlled SC-FDMA from inside

We notice from Figure 4-29 that data_in_re, data_in_im is 14-bit width (5 bits for the integer part and 9 for the fraction part), this is because the fixation problem to reduce the error which will be illustrated Chapter 6. We also notice that the data_out_re, data_out_im are 12-bit width and not 14 as the input and this is because the DFT and IFFT cores we used are generated for 12-bit width and to be used for 14-bit width we have to regenerate the cores again which need license for that. So, we generate 14-bit width data from the de-Mapper and take the most 12 bit and this is also illustrated in Chapter 6.

In our implementation, we used Xilinx IP LogiCORE Discrete Fourier Transform, the pin interface is shown Figure 4-31, the DFT module transform size is reconfigurable by the pin called size, the desired transformation size is decided as the required number of resource blocks the core indicates that it is ready to accept a new frame of data by setting RFFD high. When RFFD is high, data input may be started by setting FD_IN high for one or more cycles. Data is input via XN_RE and XN_IM. It should be provided over N cycles without interruption. Data input and output are complex and in natural order. FD_OUT signals when the core starts data output and DATA_VALID signals when data on XK_RE and XK_IM is valid.

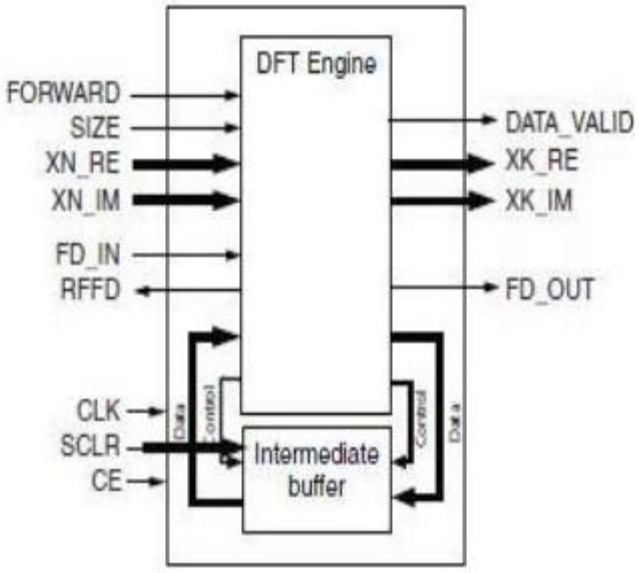


Figure 4-31: The pin interface of the used Xilinx IP LogiCORE DFT

Note that FD_IN is ignored while RFFD is low, and so FD_IN can be kept high for multiple cycles. FD_IN is accepted on the first cycle that RFFD is high, if FD_IN is set permanently high, then the core will start a new frame of data input as soon as the core is ready, this arrangement provides maximum transform

throughput. Alternatively, RFFD may be connected directly to FD_IN to achieve the same behavior. The first element of input data should be provided on the same cycle that the core starts to receive data, that is, the first cycle in which both FD_IN and RFFD are high, the IFFT module is explained more in Section 3.3.2.

The encoding of the size parameter that we used is shown in Table 4-10.

Table 4-10: The size parameter Encoding

| Size (Binary) | N | M (Radix-2) | P (Radix-3) | Q (Radix-5) | Latency $C_L$ Cycles | Period $C_T$ Cycles | Time to Process 1200 Points, µs (at 245.76 MHz) |
|---|---|---|---|---|---|---|---|
| 0 | 12 | 2 | 1 | | 75 | 62 | 25.28 |
| 1 | 24 | 3 | 1 | | 122 | 109 | 22.22 |
| 2 | 36 | 2 | 2 | | 152 | 139 | 18.90 |
| 3 | 48 | 4 | 1 | | 176 | 163 | 16.63 |

Our SC-FDMA module contains a 48-register memory, reconfigurable DFT module and 128-point IFFT module, when the mapper is ready to send data we buffer the symbols in the memory then we set the DFT size to the required size, then we start inputting symbols to the DFT and after it finishes it writes its output to the memory then we input the symbols for the IFFT. Each LTE frames consists of 6 sub frames, the third sub frame of each frame is dedicated for demodulation reference signal, in our implementation we assumed the demodulation reference signal to be all ones, and it should be improved in later designs.

Finally, the LUT utilization of the SC-FDMA block is shown in Figure 4-32.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 2620 | 0 | 53200 | 4.92 |
| LUT as Logic | 1773 | 0 | 53200 | 3.33 |
| LUT as Memory | 847 | 0 | 17400 | 4.87 |
| LUT as Distributed RAM | 176 | 0 | | |
| LUT as Shift Register | 671 | 0 | | |
| Slice Registers | 2350 | 0 | 106400 | 2.21 |
| Register as Flip Flop | 2350 | 0 | 106400 | 2.21 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 102 | 0 | 26600 | 0.38 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 4-32: SC-FDMA LUT utilization

The LUT utilization of the LTE transmitter full chain is shown in Figure 4-33.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 8179 | 0 | 53200 | 15.37 |
| LUT as Logic | 7332 | 0 | 53200 | 13.78 |
| LUT as Memory | 847 | 0 | 17400 | 4.87 |
| LUT as Distributed RAM | 176 | 0 | | |
| LUT as Shift Register | 671 | 0 | | |
| Slice Registers | 3846 | 0 | 106400 | 3.61 |
| Register as Flip Flop | 3776 | 0 | 106400 | 3.55 |
| Register as Latch | 70 | 0 | 106400 | 0.07 |
| F7 Muxes | 107 | 0 | 26600 | 0.40 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 4-33: LTE transmitter full chain LUT utilization

## 4.3 LTE Receiver PHY Block Diagram

The main target of the receiver is to retrieve the same data send before transmitter so it consists of the blocks shown in Figure 4-34.

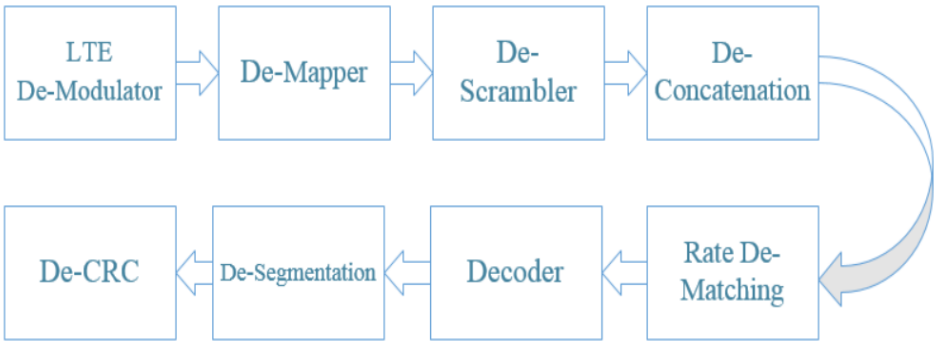


Figure 4-34: LTE Receiver full chain blocks

### 4.3.1 LTE De-modulator

LTE demodulator is the first block in the receiving chain. It is supposed to counter the effect of SC-FDMA in transmitter. SC-FDMA is implemented by the block diagram shown in Figure 4-29.

The algorithm of operation of the block can be simply explained using the flow chart in Figure 4-35.

The top controlled module of LTE De-modulator is as shown in Figure 4-36, and the pins description of the controlled module is shown in Table 4-11.

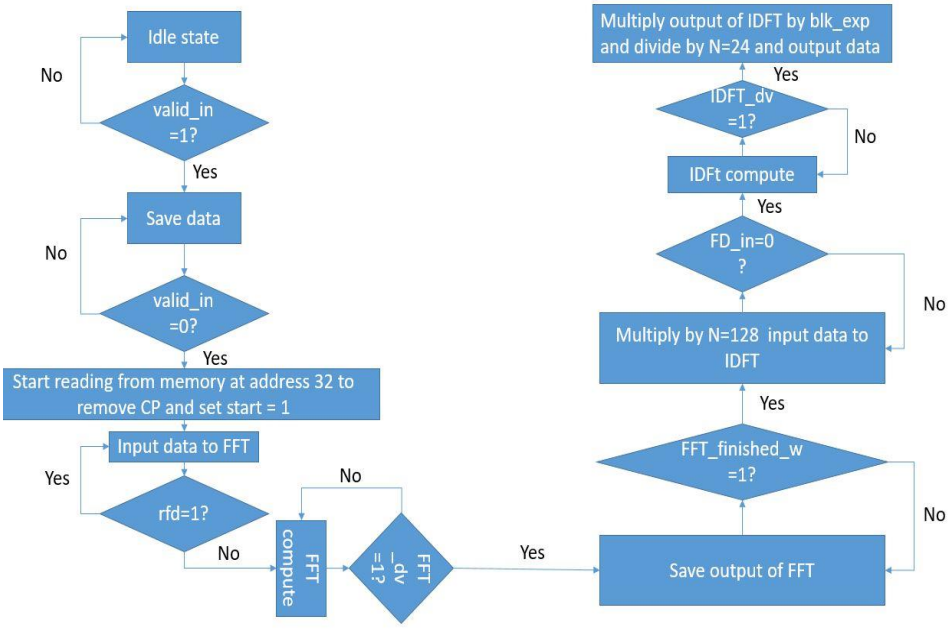


Figure 4-35: LTE De-modulator flow chart

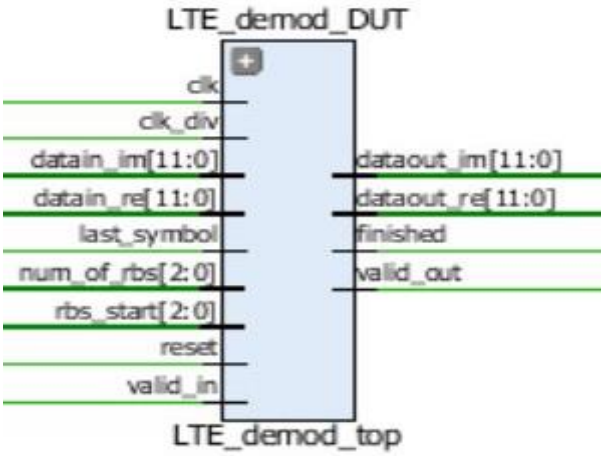


Figure 4-36: Top controlled LTE De-modulator

Table 4-11: pins description of LTE De-modulator

| Pin | Description |
|---|---|
| clk_div | Slow clock for operation of IFFT = 50 nsec |
| datain_im | The imaginary data input to the block |
| datain_re | The real data input to the block |
| dataout_im | The imaginary data output from the block |
| dataout_re | The real data output from the block |
| finished | This signal indicates that the LTE de-Modulator is ready to have data |
| last_symbol | This signal indicates that these patch of symbols are the last to be processed |
| num_of_rbs | This signal indicates the number of resource blocks mapped to the user |
| rbs_start | This signal indicates the starting resource block of the user |
| valid_in | This signal indicates that the input data is valid data |
| valid_out | This signal indicates that the output data is valid data |

Finally, the LUT utilization of the LTE De-modulator block is shown in Figure 4-37.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 2777 | 0 | 53200 | 5.22 |
| LUT as Logic | 1896 | 0 | 53200 | 3.56 |
| LUT as Memory | 881 | 0 | 17400 | 5.06 |
| LUT as Distributed RAM | 216 | 0 | | |
| LUT as Shift Register | 665 | 0 | | |
| Slice Registers | 2232 | 0 | 106400 | 2.10 |
| Register as Flip Flop | 2231 | 0 | 106400 | 2.10 |
| Register as Latch | 1 | 0 | 106400 | <0.01 |
| F7 Muxes | 90 | 0 | 26600 | 0.34 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 4-37: LTE De-modulator LUT utilization

### 4.3.2 De-Mapper

According to the 3GPP standard, LTE supports the following modulation schemes:

• QPSK •16-QAM

• 64-QAM •256-QAM

However, as explained in section 4.2.7, only QPSK modulation scheme is implemented with the following mapping described in Table 4-12.

Table 4-12: QPSK modulation scheme

| $b(i), b(i+1)$ | *I* | *Q* |
|---|---|---|
| 00 | $1/\sqrt{2}$ | $1/\sqrt{2}$ |
| 01 | $1/\sqrt{2}$ | $-1/\sqrt{2}$ |
| 10 | $-1/\sqrt{2}$ | $1/\sqrt{2}$ |
| 11 | $-1/\sqrt{2}$ | $-1/\sqrt{2}$ |

In the LTE receiver chain, data is processed in soft form (symbols), at which the hard decision is done in decoder. Hence, the de-Mapper implemented should support soft in and soft out data. At which the de-Mapper is supposed to calculate the log likelihood probability of the soft input according to this equation:

$$P(bit = 1|input) = \log_{10} \frac{d0}{d1}$$

At which d0 and d1 are the summation of distances from all symbols containing bit = 0 and the summation of distances from all symbols containing bit = 1 respectively as shown in Figure 4-38.

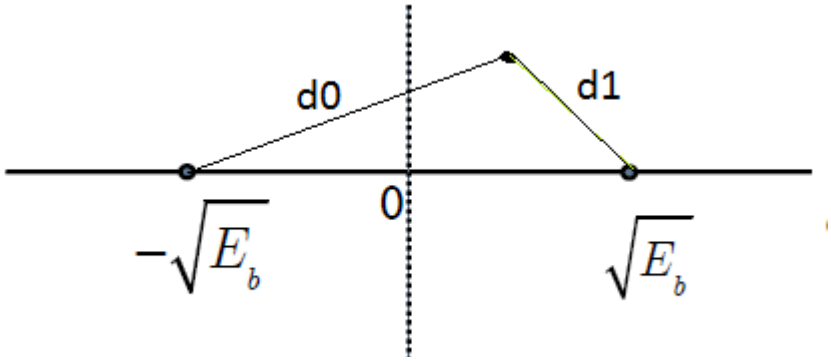


Figure 4-38: De-Mapper Soft Decision Concept

According to this equation, if d0 > d1 → $\frac{d0}{d1} > 1 \rightarrow \log_{10}\frac{d0}{d1} > 0.$

Positive Probability are more likely to be 1. As the number is more positive, it is more likely to be 1 than 0. Similarly, it can be proven that negative Probability are more likely to be 0 than 1.

For simplicity in the implemented de-Mapper, since the 1st bit is only dependent on the in-phase component and the 2nd bit is only dependent on the quadrature component. Output data will represent the same value of the symbol multiplied by either 1 or -1 with respect to the required signs of P (1) or P (0).

The top controlled module of De-Mapper is as shown in Figure 4-39, and the pins description of the controlled module is shown in Table 4-13.

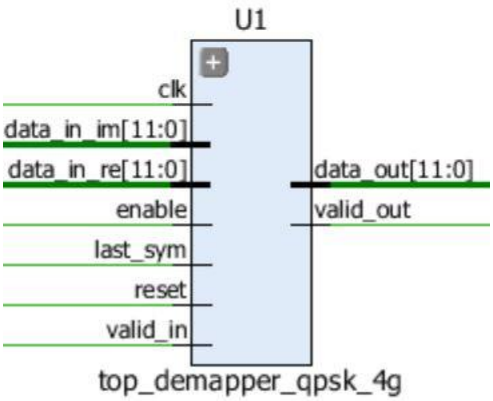


Figure 4-39: Top controlled De-Mapper

Table 4-13: pins description of De-Mapper

| Pin | Description |
|---|---|
| data_in_im | The imaginary data input to the block |
| data_in_re | The real data input to the block |
| data_out | The data output from the block |
| enable | This signal indicates that the next block (De-Scrambler) is ready to have data |
| last_sym | This signal indicates that these patch of symbols are the last to be processed |
| valid_in | This signal indicates that the input data is valid data |
| valid_out | This signal indicates that the output data is valid data |

Finally, the LUT utilization of the De-Mapper block is shown in Figure 4-40.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 81 | 0 | 53200 | 0.15 |
| LUT as Logic | 81 | 0 | 53200 | 0.15 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 65 | 0 | 106400 | 0.06 |
| Register as Flip Flop | 65 | 0 | 106400 | 0.06 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 4-40: De-Mapper LUT utilization

### 4.3.3 De-Scrambler

Typical de-Scrambling in receiver chains, should have the same implementation of the scrambling block of the transmitter. However, in case of LTE chain, the data is processed in symbols. So, bitwise operation on the symbol is not possible.

In this case, data will be multiplied by -1 (2's compliment is performed) in case the polynomial output is 1. It will pass unchanged if the polynomial output is 0. This can be illustrated by the following example in Figure 4-41.

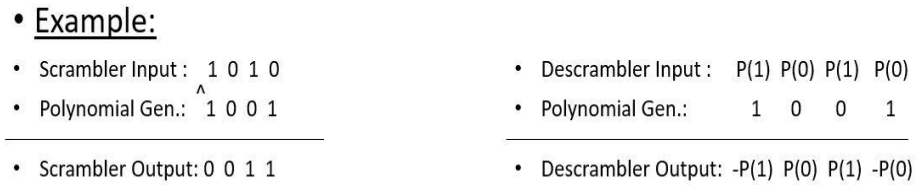


Figure 4-41: De-Scrambler operation

The top controlled module of De-Scrambler is as shown in Figure 4-42, and the pins description of the controlled module is shown in Table 4-14.

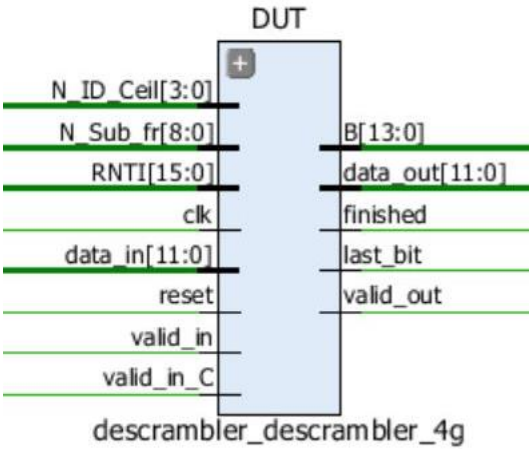


Figure 4-42: Top controlled De-Scrambler

Table 4-14: pins description of De-Scrambler

| Pin | Description |
|---|---|
| B | Number of output bits by the block |
| data_in | The soft input data to the block |
| data_out | The data output from the block |
| finished | This signal indicates that the De-Scrambler is ready to have data |
| last_bit | The last symbol outputted by the block |
| N_ID_Cell | Parameter sent by MAC layer to help in calculating scrambling polynomial |
| N_Sub_fr | Parameter sent by MAC layer to help in calculating scrambling polynomial |
| RNTI | Parameter sent by MAC layer to help in calculating scrambling polynomial |
| valid_in | This signal indicates that the input data is valid data |
| valid_in_C | valid in for C_initial parameters |
| valid_out | This signal indicates that the output data is valid data |

Finally, the LUT utilization of the De-Scrambler block is shown in Figure 4-43.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 489 | 0 | 53200 | 0.92 |
| LUT as Logic | 489 | 0 | 53200 | 0.92 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 270 | 0 | 106400 | 0.25 |
| Register as Flip Flop | 270 | 0 | 106400 | 0.25 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 4-43: De-Scrambler LUT utilization

### 4.3.4 De-Concatenation

This block is the inverse of the concatenation block which is has same function of the segmentation block with a few modifications to the method of operation to adapt to the puncturing that took place in rate matching as well as the symbol processing.

### 4.3.5 De-Segmentation

De-Segmentation has the same design & implementation as the concatenation block explained in section 2.2.4, the only difference exists in that it contains a De-CRC block inside it to contour the effect of CRC block found in the Segmentation block explained in section 4.2.2.

The top controlled module of De-Segmentation is as shown in Figure 4-44, and the pins description of the controlled module is shown in Table 4-15.

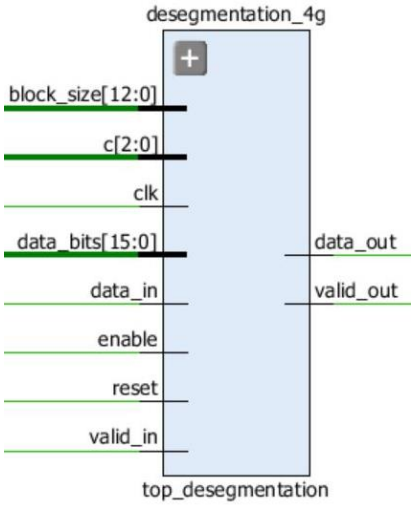


Figure 4-44: Top controlled De-Segmentation

Table 4-14: pins description of De-Segmentation

| Pin | Description |
|---|---|
| C | It is the number of code blocks computed by de-concatenation (decoder forwards it to the de-segmentation) |
| data_in | The input data to the block |
| data_out | The output data of the block |
| enable | This signal indicates that the next block (De-CRC) is ready to have data |
| finished | This signal indicates that the De-Segmentation is ready to have data |
| valid_in | This signal indicates that current data_in is valid |
| valid_out | This signal indicates that current data_out is valid |

Finally, the LUT utilization of the De-Segmentation block is shown in Figure 4-45.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 178 | 0 | 53200 | 0.33 |
| LUT as Logic | 178 | 0 | 53200 | 0.33 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 95 | 0 | 106400 | 0.09 |
| Register as Flip Flop | 95 | 0 | 106400 | 0.09 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 4-45: De-Segmentation LUT utilization

### 4.3.6 De-CRC

De-CRC has the same design & implementation as the De-CRC block explained in section 2.3.7, the only difference exists in the used polynomial where the used one here is similar to the one used in the transmitter explained in section 4.2.1.

The top controlled module of De-CRC is as shown in Figure 4-46, and the pins description of the controlled module is shown in Table 4-16.

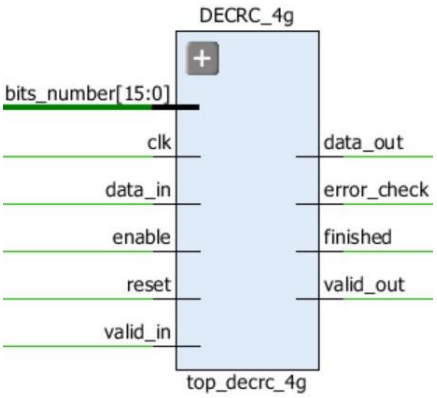


Figure 4-46: Top controlled De-CRC

Table 4-16: pins description of De-CRC

| Pin | Description |
|---|---|
| bits_number | This signal indicates the total number of bits entering the block |
| data_in | The input data to the block |
| data_out | The output data of the block |
| error_check | This signal indicates the result of comparison between the generated bits from the block with the last bits received |
| finished | This signal indicates that the De-CRC is ready to have data |
| valid_in | This signal indicates that current data_in is valid |
| valid_out | This signal indicates that current data_out is valid |

Finally, the LUT utilization of the De-CRC block is shown in Figure 4-47.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 131 | 0 | 53200 | 0.25 |
| LUT as Logic | 131 | 0 | 53200 | 0.25 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 93 | 0 | 106400 | 0.09 |
| Register as Flip Flop | 93 | 0 | 106400 | 0.09 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 4-47: De-CRC LUT utilization

CHAPTER 5

# GSM Chain

## 5.1 Introduction

GSM is a standard developed by the ETSI to describe the protocols for 2G digital cellular networks used by mobile phones, first deployed in Finland in December 1991. As of 2014, it has become the de facto global standard for mobile communications – with over 90% market share, operating in over 219 countries and territories.

2G networks developed as a replacement for 1G analog cellular networks, and the GSM standard originally described as a digital, circuit-switched network optimized for full duplex voice telephony. This expanded over time to include data communications, first by circuit-switched transport, then by packet data transport via GPRS and EDGE.

## 5.2 GSM Transmitter PHY Block Diagram

Transmitter of GSM consists of several blocks as shown in Figure 5-1. In the following sub-sections, each block of the chain is explained in more details _ illustrating its basic idea, showing its interfaces, connections, inputs & outputs & presenting its LUT utilization_.

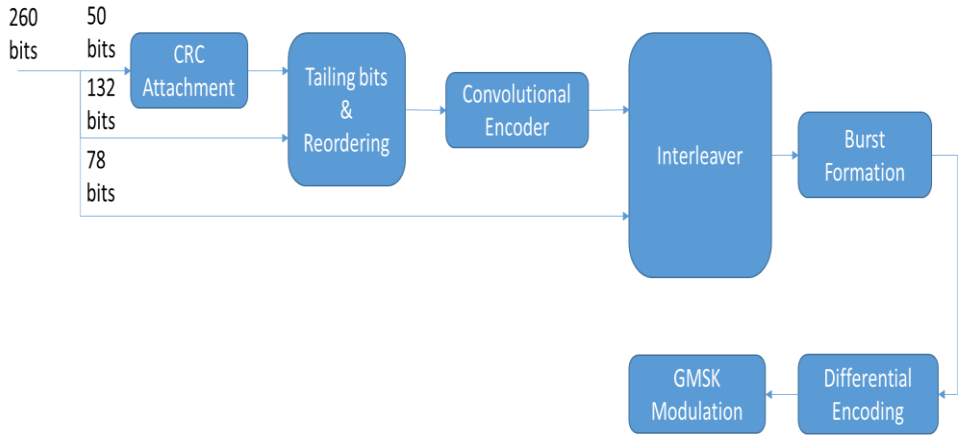


Figure 5-1: GSM Transmitter full chain blocks

### 5.2.1 CRC Attachment, Bit Tailing & Reordering

#### 5.2.1.1 CRC

CRC process is provided on transport blocks for error detection in which the entire block is used to calculate the CRC parity bits for each transport block. Instead of adding just one bit to a block of data, several bits are added. The size of the CRC is 3 bits. These parity bits are added to the 50 bits, known as class 1 for FR, according to a degenerate (shortened) cyclic code (53, 50), using the generator polynomial:

$$g(D) = D^3 + D + 1$$

#### 5.2.1.2 Bit Tailing & Reordering

The output of CRC (53 bits) is reordered with the next 132 bits of the information bits from the 260 bits resulting from source encoding. After Tailing bits are attached, according to the next equation:

$$u(k) = d(2k), \quad \text{for } k = 0, 1, ..., 90$$

$$u(184-k) = d(2k+1), \quad \text{for } k = 0, 1, ..., 90$$

$$u(91+k) = p(k), \quad \text{for } k = 0, 1, 2$$

$$u(k) = 0, \quad \text{for } k = 185, 186, 187, 188 \text{ (tail bits)}$$

where, u(k) denotes the output of bit tailing and reordering block, d(k) denotes information bits from source coding, p(k) denotes parity bits attached by CRC. The rest of the information bits pass without processing or reordering.

The top controlled module of CRC Attachment, Bit Tailing & Reordering is as shown in Figure 5-2, and the pins description of the controlled module is shown in Table 5-1.

Finally, the LUT utilization of the CRC Attachment, Bit Tailing & Reordering block is shown in Figure 5-3.

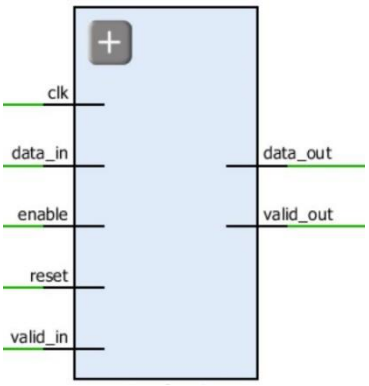


Figure 5-2: Top controlled CRC Attachment, Bit Tailing & Reordering

Table 5-1: pins description of CRC Attachment, Bit Tailing & Reordering

| Pin | Description |
|---|---|
| clk | Operation clock of the block. It should be set to 6.4 µsec to achieve the output rate according to standard. |
| reset | Ensures that all registers are set to zero at the start of the operation. |
| data_in | The input data from the source encoding, its length is always 260 bits for every 20 msec speech. |
| enable | It indicates that the next block is ready to receive new data from this block. |
| valid_in | A signal that indicates that the input is valid for processing. |
| data_out | The output data from the block. Its length should be equal to 267 bits. |
| valid_out | A signal that indicates that the output is valid for processing. |

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 235 | 0 | 53200 | 0.44 |
| LUT as Logic | 203 | 0 | 53200 | 0.38 |
| LUT as Memory | 32 | 0 | 17400 | 0.18 |
| LUT as Distributed RAM | 32 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 100 | 0 | 106400 | 0.09 |
| Register as Flip Flop | 100 | 0 | 106400 | 0.09 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 20 | 0 | 26600 | 0.08 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 5-3: CRC Attachment, Bit Tailing & Reordering LUT utilization

### 5.2.2 Encoder

Coding is the process of adding redundant data to give the receiver the ability for FER (correcting received data without need for retransmission). In GSM standard, a convolutional code is introduced with constraint length 5 and coding rate ½. The block diagram of the encoder is shown in Figure 5-4. Input to the encoder according to the standard is 189 bits while the other 78 bits pass un-coded to make the length of the output 456 bits.

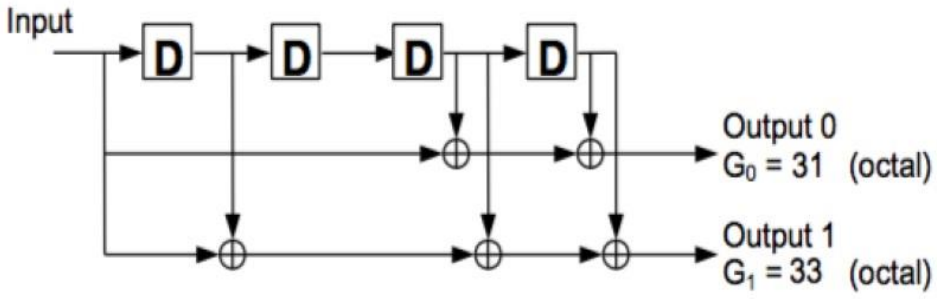


Figure 5-4: Convolutional Encoder block diagram

The top controlled module of Encoder is as shown in Figure 5-5, and the pins description of the controlled module is shown in Table 5-2.

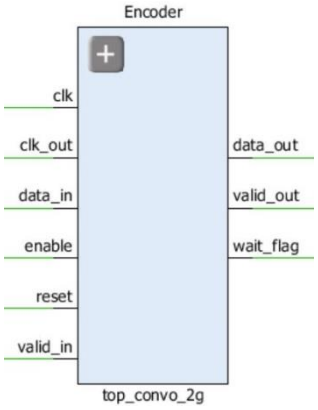


Figure 5-5: Top controlled Encoder

Finally, the LUT utilization of the Encoder block is shown in Figure 5-6.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 87 | 0 | 53200 | 0.16 |
| LUT as Logic | 85 | 0 | 53200 | 0.16 |
| LUT as Memory | 2 | 0 | 17400 | 0.01 |
| LUT as Distributed RAM | 2 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 59 | 0 | 106400 | 0.06 |
| Register as Flip Flop | 59 | 0 | 106400 | 0.06 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 5-6: Encoder LUT utilization

Table 5-2: pins description of Encoder

| Pin | Description |
|---|---|
| clk | Input clock of the block. It should be set to 6.4 µsec to achieve the output rate according to standard. |
| clk_out | Output clock of the block. It should be set to half the input clock for correct operation of the block, 3.2 µsec. |
| reset | Ensures that all registers are set to zero at the start of the operation. |
| data_in | The input data from the source encoding. |
| enable | It indicates that the next block is ready to receive new data from this block. |
| valid_in | A signal that indicates that the input is valid for processing. |
| data_out | The output data from the block. |
| valid_out | A signal that indicates that the output is valid for processing. |
| wait_flag | A signal that is set to high during the period, at which valid_out is high to indicate that specific bits are not valid for processing to ensure that output bit length is 456 bits. |

### 5.2.3 Interleaver

Interleaving is a way to re-arrange data in a non-contiguous way to make it stand burst errors. These types of errors can destroy many bits in a row and make it hard to recover using FEC coding, since these expects the errors to be more uniformly distributed. This method is popular because it is a less complex and cheaper way to handle burst errors than directly increasing the power of the error correction scheme and interleaving cause increasing the performance of decoding as shown in Table 5-3.

Table 5-3: Without VS with interleaving

| Without interleaving | With interleaving |
|---|---|
| Transmitted Bits :<br>b0 b1 b2 b3 b4 b5 b6 b7 b8<br>Received Bits :<br>b0 b1 b2 b3 x x x b7 b8<br>(x indicates to error in bit) | Transmitted Bits :<br>b0 b1 b2 b3 b4 b5 b6 b7 b8<br>Interleaved Bits :<br>b0 b3 b6 b1 b4 b7 b2 b5 b8<br>Received Bits :<br>b0 b3 b6 b1 x x x b5 b8<br>De-interleaved Bits:<br>b0 b1 x b4 b5 b6 x b8 |
| Burst errors<br>hard to recover | Distributed errors |

The main disadvantage of interleaving techniques is increasing latency because the entire interleaved block must be received before the packets can be decoded.

In GSM standard, Interleaver receives 465 bits which are written in a form of 8x57 matrix row by row. Then, it is read column by column as shown in Figure 5-7. Hence, the output data that should be read are 1, 9, 17…449, 2, 10 … 450…

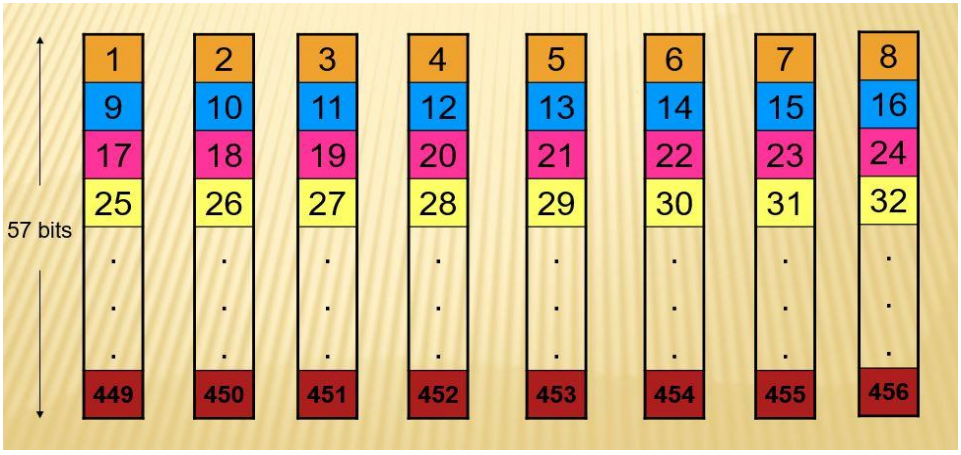


Figure 5-7: Data Matrix

According to standard, data should be mapped to burst in form of a column from 2 different segments to decrease the losses of packets to 12.5%. However, for simplicity, it is implemented that data is mapped to burst in the following form:

1st & 5th – 2nd & 6th -3rd & 7th -4th & 8th.

The top controlled module of Interleaver is as shown in Figure 5-8, and the pins description of the controlled module is shown in Table 5-4.

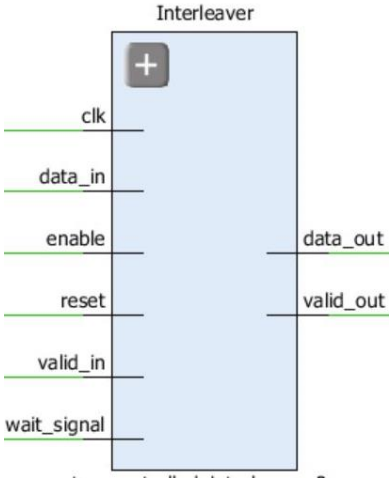


Figure 5-8: Top controlled Interleaver

Finally, the LUT utilization of the Interleaver block is shown in Figure 5-9.

Table 5-4: pins description of Interleaver

| Pin | Description |
|---|---|
| clk | Operation clock of the block. It should be set to 3.2 µsec. |
| reset | Ensures that all registers are set to zero at the start of the operation. |
| data_in | The input data from the encoder block |
| enable | It indicates that the next block is ready to receive new data from this block. |
| valid_in | A signal that indicates that the input is valid for processing. |
| wait_signal | A signal that indicates that this input is not to saved. |
| data_out | The output data from the block. |
| valid_out | A signal that indicates that the output is valid for processing. |

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 174 | 0 | 53200 | 0.33 |
| LUT as Logic | 158 | 0 | 53200 | 0.30 |
| LUT as Memory | 16 | 0 | 17400 | 0.09 |
| LUT as Distributed RAM | 16 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 44 | 0 | 106400 | 0.04 |
| Register as Flip Flop | 44 | 0 | 106400 | 0.04 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 19 | 0 | 26600 | 0.07 |
| F8 Muxes | 2 | 0 | 13300 | 0.02 |

Figure 5-9: Interleaver LUT utilization

### 5.2.4 Burst formation and multiplexing

This block is essentially responsible for the time division. It receives 114 bit from Interleaver, then it starts forming the burst as mentioned in Table 5-5. It is implemented as in Figure 5-10 and is operated according the flow chart shown in Figure 5-11.

Table 5-5: Burst formation

| Bit Number (BN) | Length of field | Contents of field |
|---|---|---|
| 0 - 2 | 3 | tail bits |
| 3 - 60 | 58 | encrypted bits (e0 . e57) |
| 61 - 86 | 26 | training sequence bits |
| 87 - 144 | 58 | encrypted bits (e58 . e115) |
| 145 - 147 | 3 | tail bits |
| 148 - 156 | 8,25 | guard period (bits) |

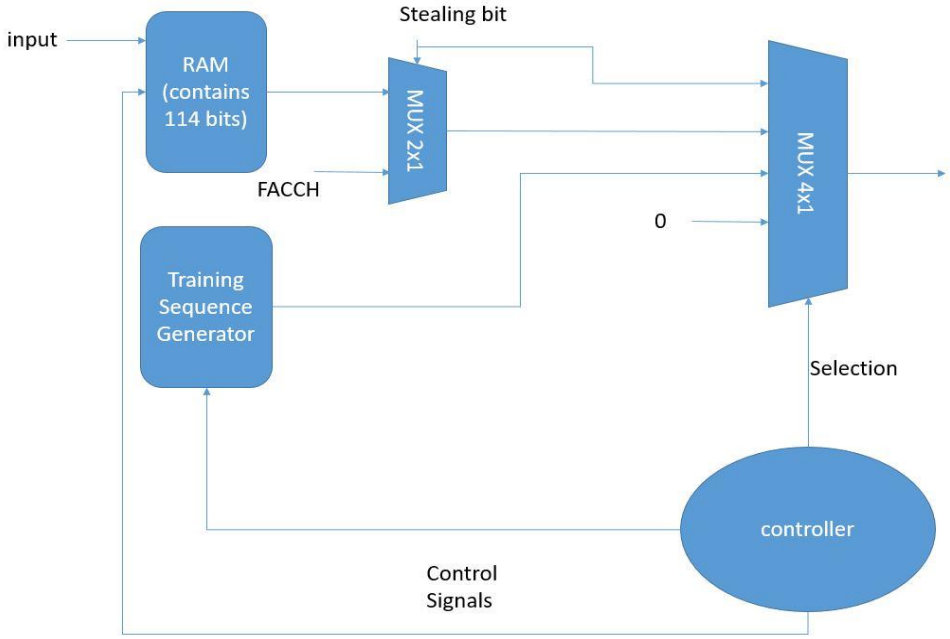


Figure 5-10: Burst formation and multiplexing Block diagram

It should be noted that if steal flag is 1, FACCH is sent instead of the data segment in the burst. Also, 1250 is chosen according to the clock of the block (3.2 µsec). Hence, 1250*3.2µsec = 4 msec and 156.25*3.2µsec = 0.5 msec.

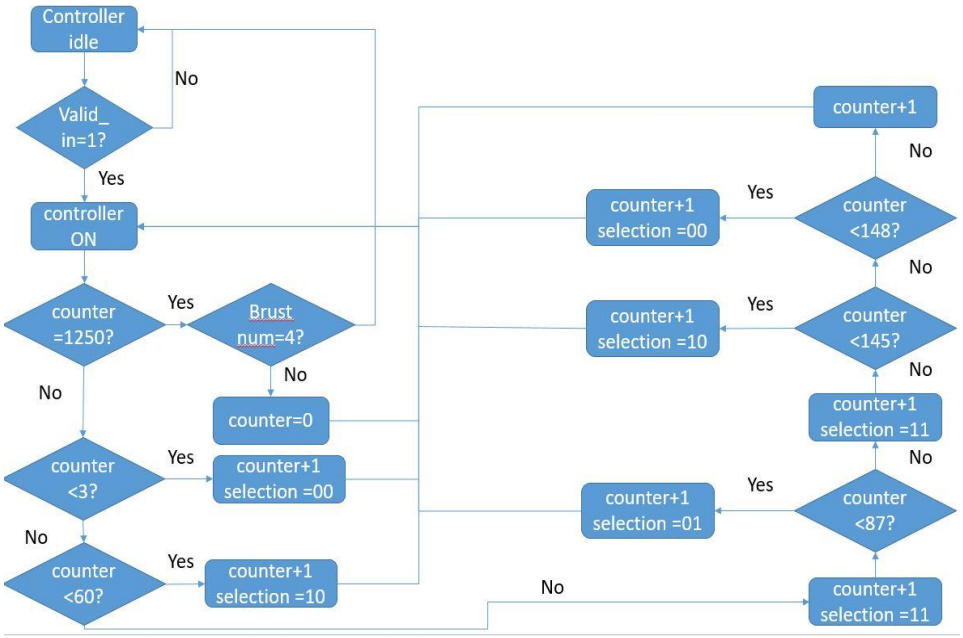


Figure 5-11: Burst formation and multiplexing Flow chart

The top controlled module of Burst formation and multiplexing is as shown in Figure 5-12, and the pins description of the controlled module is shown in Table 5-6.

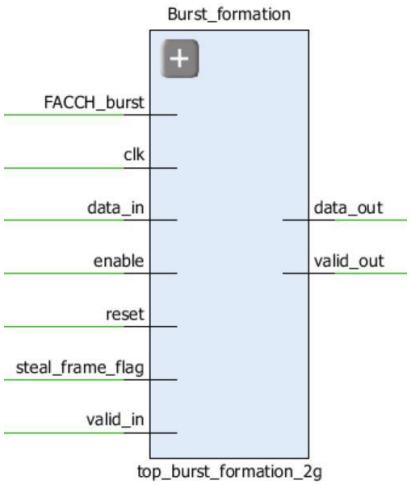


Figure 5-12: Top controlled Burst formation and multiplexing

Table 5-6: pins description of Burst formation and multiplexing

| Pin | Description |
|---|---|
| clk | Operation clock of the block. It should be set to 6.4 µsec to achieve the output rate according to standard. |
| reset | Ensures that all registers are set to zero at the start of the operation. |
| data_in | The input data from the source encoding, its length is always 260 bits for every 20 msec speech. |
| enable | It indicates that the next block is ready to receive new data from this block. |
| valid_in | A signal that indicates that the input is valid for processing. |
| FACCH | Input from MAC layer |
| Stealing_bit | Input from MAC layer to indicate whether TrCH is sent on burst or FACCH |
| data_out | The output data from the block. Its length should be equal to 267 bits. |
| valid_out | A signal that indicates that the output is valid for processing. |

Finally, the LUT utilization of the Burst formation and multiplexing block is shown in Figure 5-13.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 144 | 0 | 53200 | 0.27 |
| LUT as Logic | 128 | 0 | 53200 | 0.24 |
| LUT as Memory | 16 | 0 | 17400 | 0.09 |
| LUT as Distributed RAM | 16 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 46 | 0 | 106400 | 0.04 |
| Register as Flip Flop | 46 | 0 | 106400 | 0.04 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 9 | 0 | 26600 | 0.03 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 5-13: Burst formation and multiplexing LUT utilization

### 5.2.5 Differential Encoding & GMSK Modulation

In GMSK Modulation, data is supposed to pass through a differential encoder then it is converted to NRZ form where it passes through a Gaussian filter to shape the data with continuous phase then it passes through RF modulation as shown in Figure 5-14.

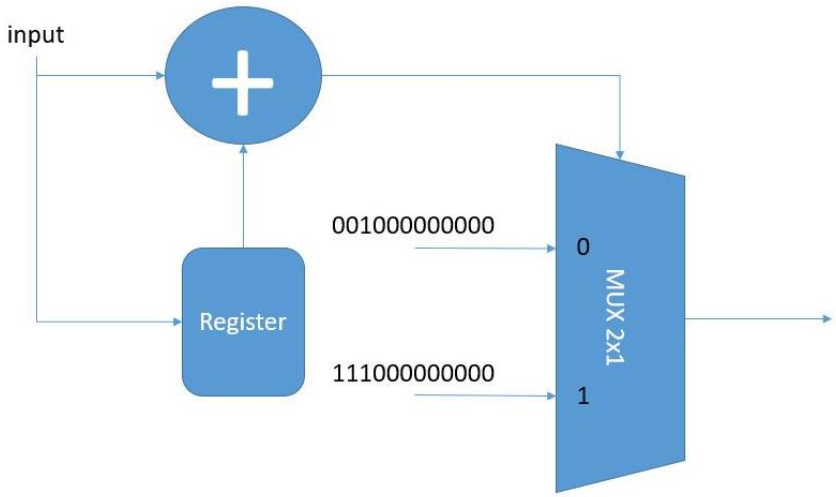


Figure 5-14: Modulation Block diagram

In the implemented system, data is outputted with NRZ form since the Gaussian filter changed signal to analog which is out of the scope of our project. Differential encoding block should perform the function below:

$$d_i = d_i \oplus d_{i-1}, (d_i \in \{0,1\})$$

Then it is converted to NRZ with this formula:

$$\alpha = 1\text{-}2*d_i$$

The top controlled module of Modulation is as shown in Figure 5-15, and the pins description of the controlled module is shown in Table 5-7.

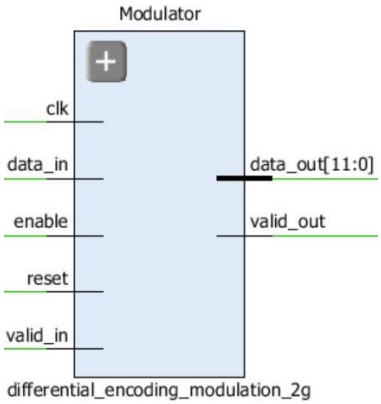


Figure 5-15: Top controlled Modulation

Table 5-7: pins description of Modulation

| Pin | Description |
|---|---|
| clk | Operation clock of the block. |
| reset | Ensures that all registers are set to zero at the start of the operation. |
| data_in | The input data from burst formation and multiplexing. |
| enable | It indicates that the next block is ready to receive new data from this block. |
| valid_in | A signal that indicates that the input is valid for processing. |
| data_out | The output data from the block. It is a 12-bit bus symbol. |
| valid_out | A signal that indicates that the output is valid for processing. |

Finally, the LUT utilization of the Modulation block is shown in Figure 5-16.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 2 | 0 | 53200 | <0.01 |
| LUT as Logic | 2 | 0 | 53200 | <0.01 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 1 | 0 | 106400 | <0.01 |
| Register as Flip Flop | 1 | 0 | 106400 | <0.01 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 5-16: Modulation LUT utilization

The LUT utilization of the GSM transmitter full chain is shown in Figure 5-17.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 647 | 0 | 53200 | 1.22 |
| LUT as Logic | 581 | 0 | 53200 | 1.09 |
| LUT as Memory | 66 | 0 | 17400 | 0.38 |
| LUT as Distributed RAM | 66 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 241 | 0 | 106400 | 0.23 |
| Register as Flip Flop | 241 | 0 | 106400 | 0.23 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 54 | 0 | 26600 | 0.20 |
| F8 Muxes | 3 | 0 | 13300 | 0.02 |

Figure 5-17: GSM transmitter full chain LUT utilization

## 5.3 GSM Receiver PHY Block Diagram

Receiver of GSM consists of several blocks as shown in Figure 5-18. In the following sub-sections, each block of the chain is explained in more details _ illustrating its basic idea, showing its interfaces, connections, inputs & outputs & presenting its LUT utilization_.

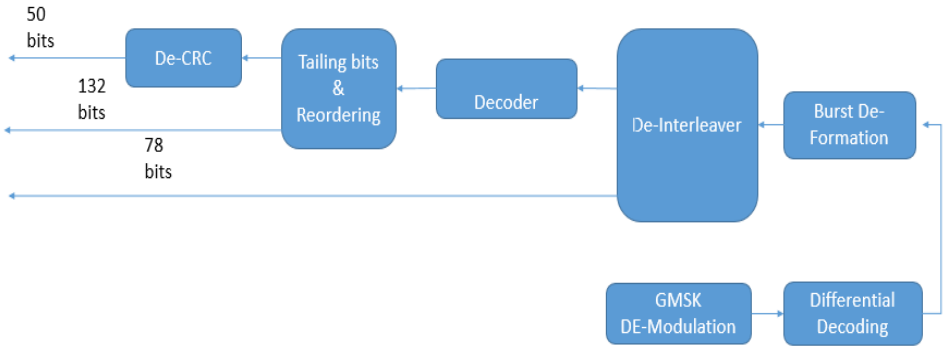


Figure 5-18: GSM Receiver full chain blocks

### 5.3.1 GMSK De-Modulation & Differential Decoding

Data is modulated and reshaped with the Gaussian filter which is out of the scope of this project. Then differential decoding is performed as in Figure 5-19. It should be noted that error is accumulative in differential decoding & symbols are converted to hard bits through the sign of the symbol.

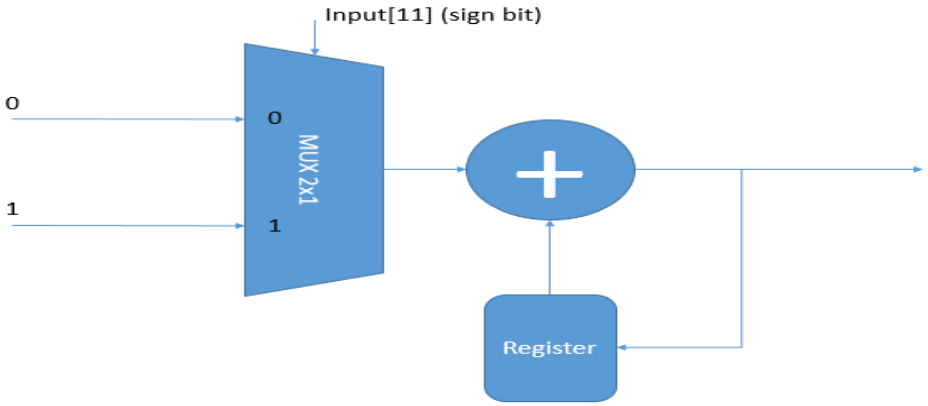


Figure 5-19: Differential decoding Block diagram

The top controlled module of De-Modulation is as shown in Figure 5-20, and the pins description of the controlled module is shown in Table 5-8.

Finally, the LUT utilization of the De-Modulation block is shown in Figure 5-21.

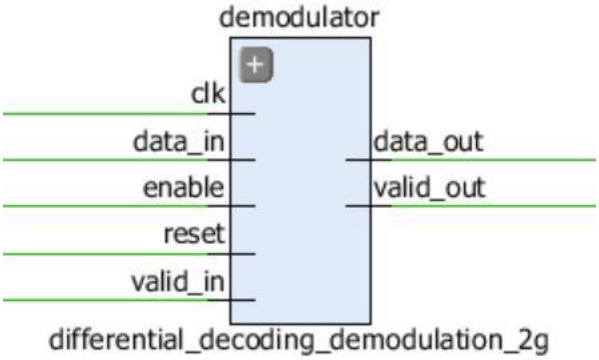


Figure 5-20: Top controlled De-Modulation

Table 5-8: pins description of De-Modulation

| Pin | Description |
|---|---|
| clk | Operation clock of the block. |
| reset | Ensures that all registers are set to zero at the start of the operation. |
| data_in | The input data |
| enable | It indicates that the next block is ready to receive new data from this block. |
| valid_in | A signal that indicates that the input is valid for processing. |
| data_out | The output data from the block. It is a 12-bit bus symbol. |
| valid_out | A signal that indicates that the output is valid for processing. |

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 2 | 0 | 53200 | <0.01 |
| LUT as Logic | 2 | 0 | 53200 | <0.01 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 1 | 0 | 106400 | <0.01 |
| Register as Flip Flop | 1 | 0 | 106400 | <0.01 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 5-21: De-Modulation LUT utilization

### 5.3.2 Burst De-formation

This block performs the inverse function of the Burst formation and multiplexing block in the transmitter. The top controlled module of Burst De-formation is as shown in Figure 5-22, and the pins description of the controlled module is shown in Table 5-9.

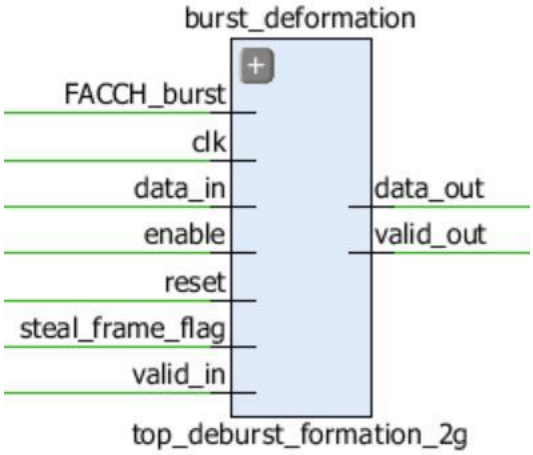

Figure 5-22: Top controlled Burst De-formation

Table 5-9: pins description of Burst De-formation

| Pin | Description |
|---|---|
| clk | Operation clock of the block. It should be set to 6.4 µsec to achieve the output rate according to standard. |
| reset | Ensures that all registers are set to zero at the start of the operation. |
| data_in | The input data to the block. |
| enable | It indicates that the next block is ready to receive new data from this block. |
| valid_in | A signal that indicates that the input is valid for processing. |
| FACCH_burst | Input from MAC layer |
| Steal_frame_flag | Input to indicate whether TrCH is sent on burst or FACCH from the transmitter |
| data_out | The output data from the block. |
| valid_out | A signal that indicates that the output is valid for processing. |

Finally, the LUT utilization of the Burst De-formation block is shown in Figure 5-23.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 78 | 0 | 53200 | 0.15 |
| LUT as Logic | 78 | 0 | 53200 | 0.15 |
| LUT as Memory | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 37 | 0 | 106400 | 0.03 |
| Register as Flip Flop | 37 | 0 | 106400 | 0.03 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 0 | 0 | 26600 | 0.00 |
| F8 Muxes | 0 | 0 | 13300 | 0.00 |

Figure 5-23: Burst De-formation LUT utilization

### 5.3.3 De-Interleaver

This block counters the effect caused by the Interleaver in the transmitter in order to deliver the data successfully to the decoder. The top controlled module of De-Interleaver is as shown in Figure 5-24, and the pins description of the controlled module is shown in Table 5-10.

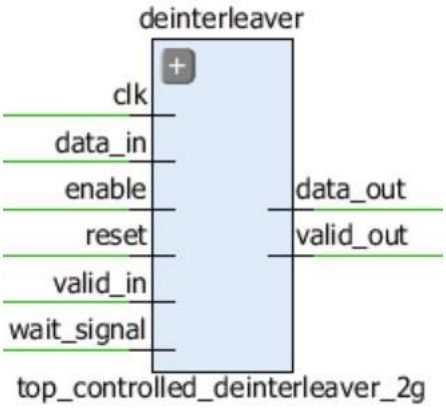


Figure 5-24: Top controlled De-Interleaver

Table 5-10: pins description of De-Interleaver

| Pin | Description |
|---|---|
| clk | Operation clock of the block. It should be set to 3.2 µsec. |
| reset | Ensures that all registers are set to zero at the start of the operation. |
| data_in | The input data |
| enable | It indicates that the next block is ready to receive new data from this block. |
| valid_in | A signal that indicates that the input is valid for processing. |
| wait_signal | A signal that indicates that this input is not to saved. |
| data_out | The output data from the block. |
| valid_out | A signal that indicates that the output is valid for processing. |

Finally, the LUT utilization of the De-Interleaver block is shown in Figure 5-25.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs* | 179 | 0 | 53200 | 0.34 |
| LUT as Logic | 163 | 0 | 53200 | 0.31 |
| LUT as Memory | 16 | 0 | 17400 | 0.09 |
| LUT as Distributed RAM | 16 | 0 | | |
| LUT as Shift Register | 0 | 0 | | |
| Slice Registers | 48 | 0 | 106400 | 0.05 |
| Register as Flip Flop | 48 | 0 | 106400 | 0.05 |
| Register as Latch | 0 | 0 | 106400 | 0.00 |
| F7 Muxes | 20 | 0 | 26600 | 0.08 |
| F8 Muxes | 2 | 0 | 13300 | 0.02 |

Figure 5-25: De-Interleaver LUT utilization

### 5.3.4 Decoder

Viterbi Decoder is the reverse block of the convolutional encoder. The block design is the same as that described in section 2.3.5, the only differences exist in that the used convolutional encoder has K=5 so as the Viterbi decoder here which decreases the number of the states to 16 other than 256.

### 5.3.5 Re-ordering & De-CRC

The block design is similar to that of the transmitter explained in section 5.2.1.

CHAPTER 6

# Verification Methodology

## 6.1 Functional Verification

Functional Verification is very important to check that HDL implementations outputs are identical to expected outputs, we implemented MATLAB models for all transmitter and receiver blocks to use as a reference models.

Verification methodology is based on equivalence checking between HDL Model and MATLAB Model, we setup testing framework consists of Verilog testbench, MATLAB testbench and Perl script.

Figure 6-1 shows the functional verification procedure; first Verilog testbench generates random input bits and store it in a file and also save the output data bits from all blocks in different files, MATLAB testbench read the input file and also save the output data bits from all blocks in different files, Perl script compare these files and generate output comparison file.

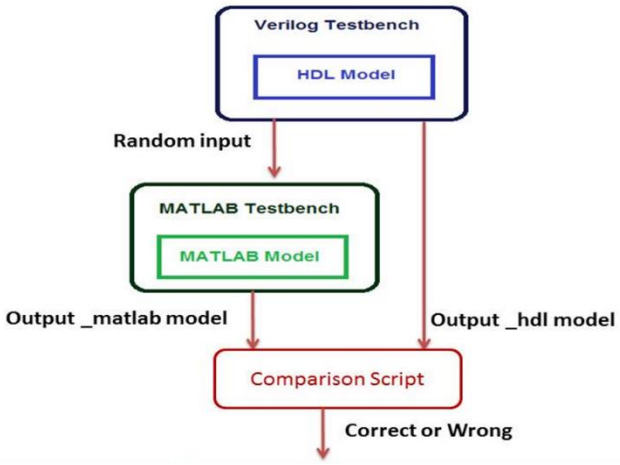


Figure 6-1: Functional Verification procedure

To cover different cases, we change number of bits and also number of successive frames to check that every block can reset its state after frame finished and process next frame correctly.

## 6.2 Sources of Noise

The receiver of any communication chain should counter what happened in the transmitter, in a way to be able to correct various error types. Error can be introduced by interference, lack of synchronization, fixation error…etc. However, in the scope of this project, only 3 main contributors of noise are considered.

### 6.2.1 Noise

Noise can be defined as an unwanted signal that is introduced to the receiver. This noise can be internal which is the thermal noise of the electric components or external. Since, it is a random signal

It is modelled using AWGN. This error can be partially corrected using decoder block in the receiver.

For Wi-Fi chain, Figure 6-2 shows the SNR vs BER for the modulation scheme "BPSK" while Figure 6-3 shows the SNR vs BER for the modulation scheme "QPSK".

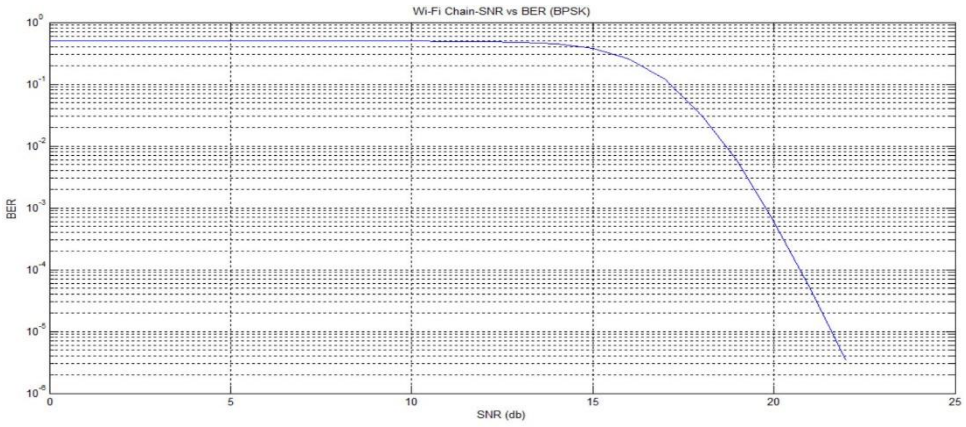


Figure 6-2: Wi-Fi SNR vs BER for BPSK "Noise error"

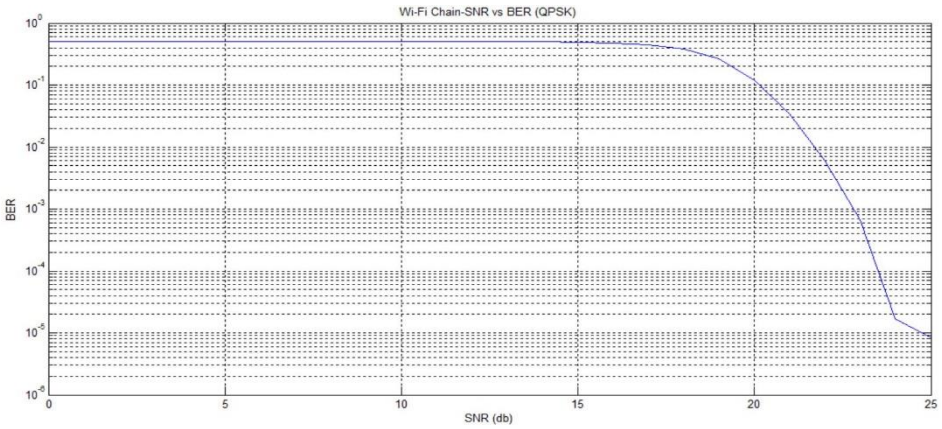


Figure 6-3: Wi-Fi SNR vs BER for QPSK "Noise error"

For 2G chain, Figure 6-4 shows the SNR vs BER for the modulation scheme "GMSK", for 3G chain, Figure 6-5 shows the SNR vs BER for the modulation scheme "BPSK" & finally for LTE chain, Figure 6-6 shows the SNR vs BER for the modulation scheme "QPSK".

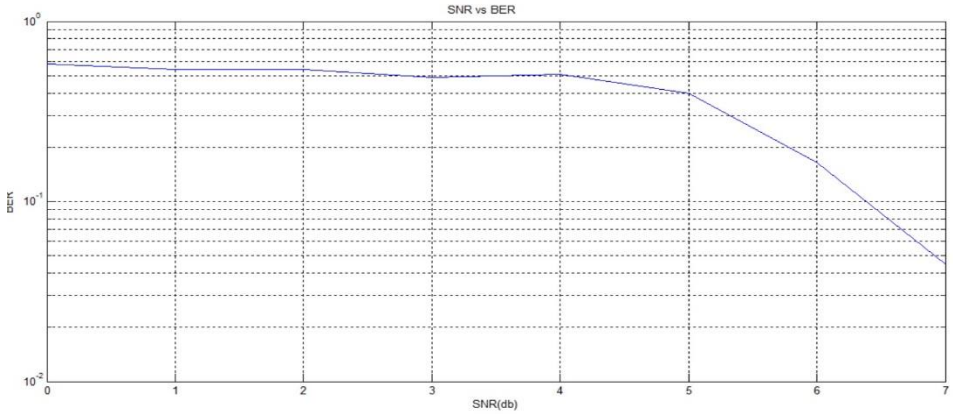


Figure 6-4: 2G SNR vs BER "Noise error"

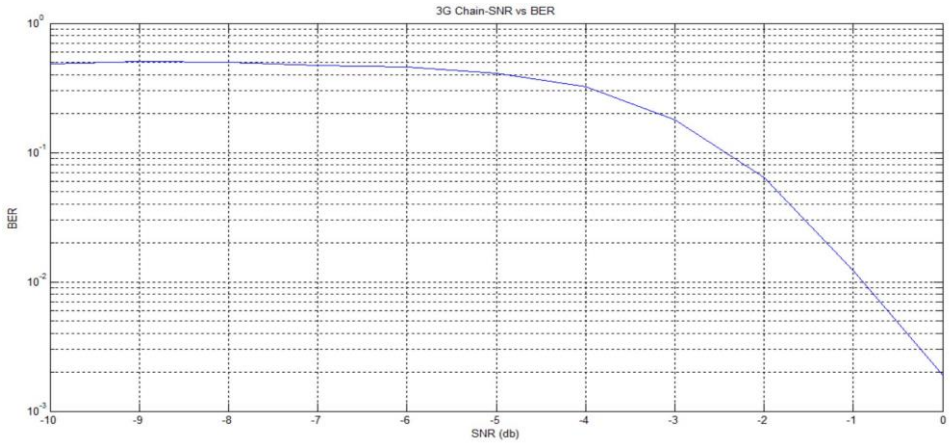


Figure 6-5: 3G SNR vs BER for BPSK "Noise error"

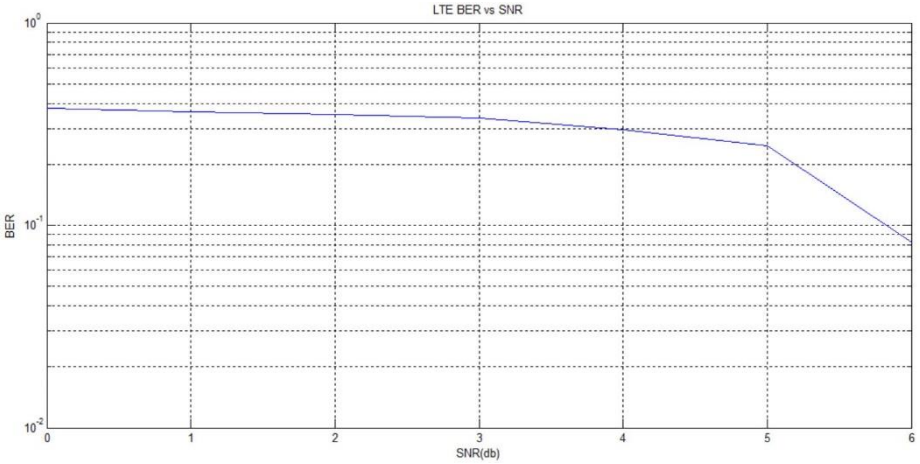


Figure 6-6: LTE SNR vs BER for QPSK "Noise error"

### 6.2.2 Channel Effect

Channel can be defined as the physical medium of transmission. It can be either wired or wireless. However, in all mobile systems, channel of concern is the wireless model. Wireless channels suffer from 3 main issues: Path loss, Shadowing, Multi-path fading. Hence, channel can be divided into slow & fast fading in time according to the Doppler spread. It can also be divided into flat and frequency selective channel according to the coherence bandwidth. This introduces a major error for the received signal.

The errors introduced by channel effect can be compensated if the signal is divided by the channel. Hence, to compensate the channel effect, a channel estimation block is required.

For 2G chain, Figure 6-7 shows that 26 of every fame are specified to channel estimation. Hence, channel estimation block is implemented & the result (SNR vs BER) is shown in Figure 6-8 while in Figure 6-9, the result is shown for an implemented theoretical equalizer block.


Figure 6-7: 2G frame

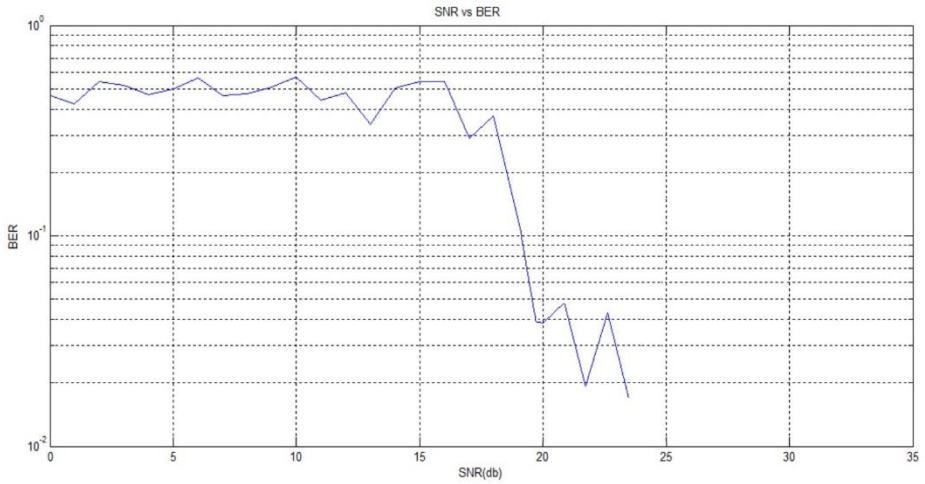


Figure 6-8: 2G SNR vs BER for Channel Estimation Block "channel error"

For 3G chain, Implementation of channel estimation is 3G is relatively impossible since the transmitter chain contains no pilots.

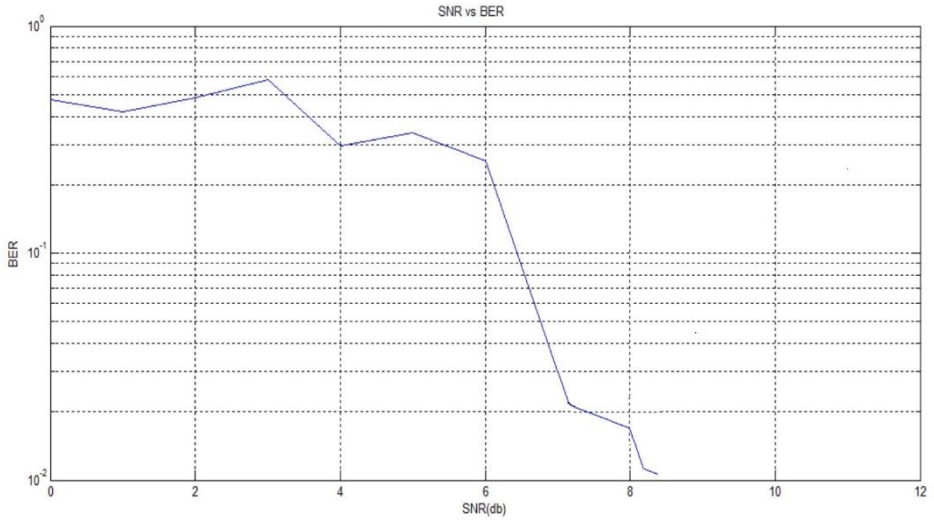


Figure 6-9: 2G SNR vs BER for Theoretical Equalizer Block "channel error"

For LTE chain, A reference signal (chosen 1 for simplicity), is sent over all relevant sub-carriers every 7 sub-frames, this is called block pilot. Hence, the channel estimator works fine at slow fading channel.

Channel estimation block is implemented & also, theoretical equalizer block is implemented and the result (SNR vs BER) is shown in Figures 6-10, 6-11 respectively for flat fading channel and in Figures 6-12, 6-13 respectively for frequency selective channel.

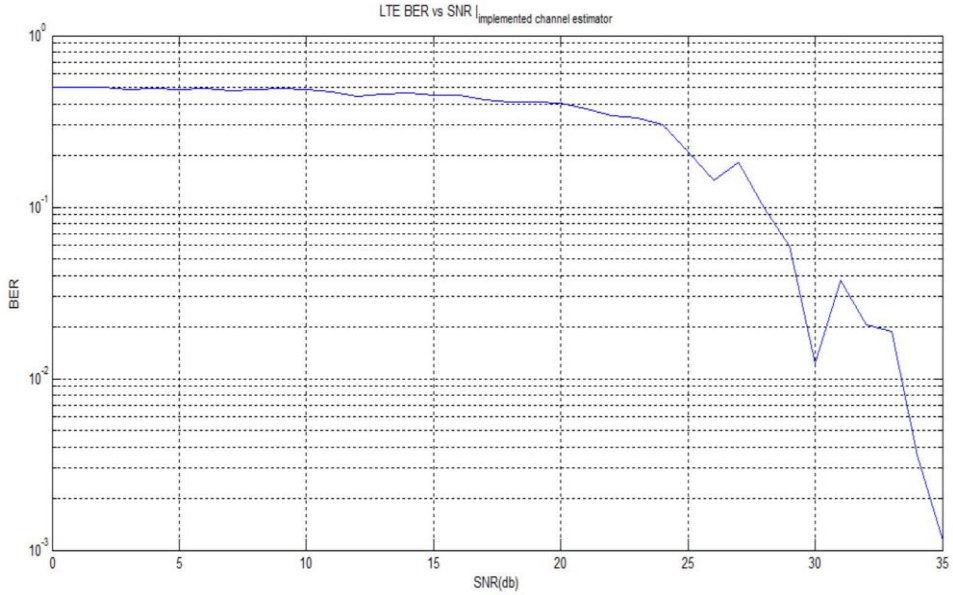


Figure 6-10: LTE SNR vs BER for Channel Estimation Block "flat channel error"

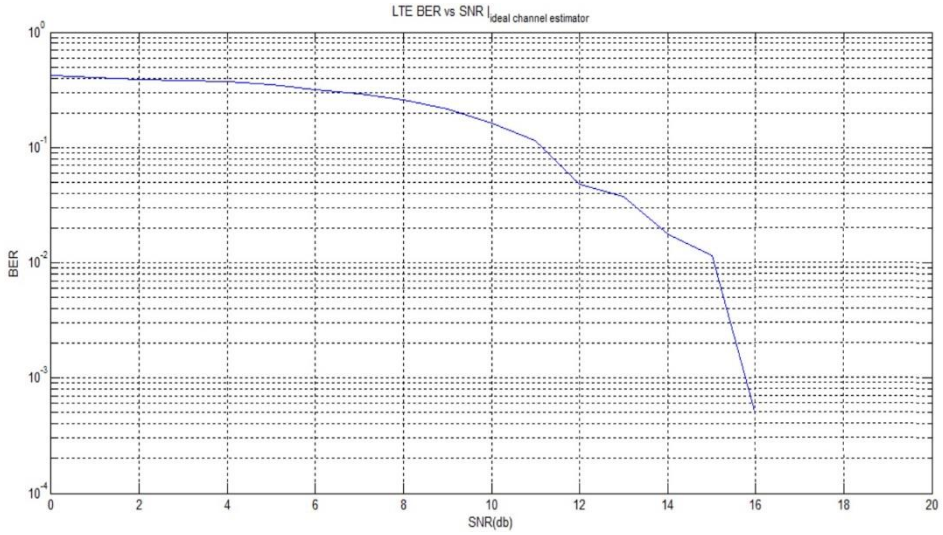


Figure 6-11: LTE SNR vs BER for Theoretical Equalizer Block "flat channel error"

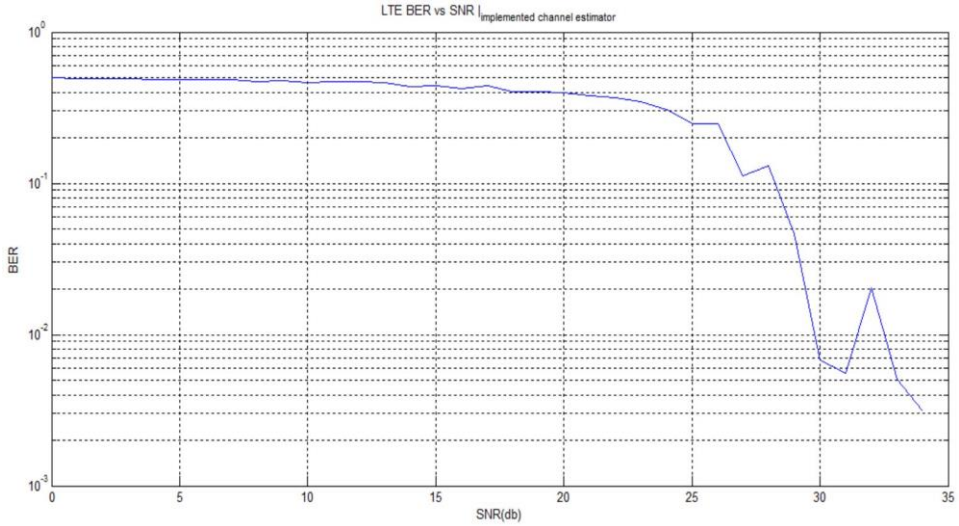


Figure 6-12: LTE SNR vs BER for Channel Estimation Block "selective channel error"

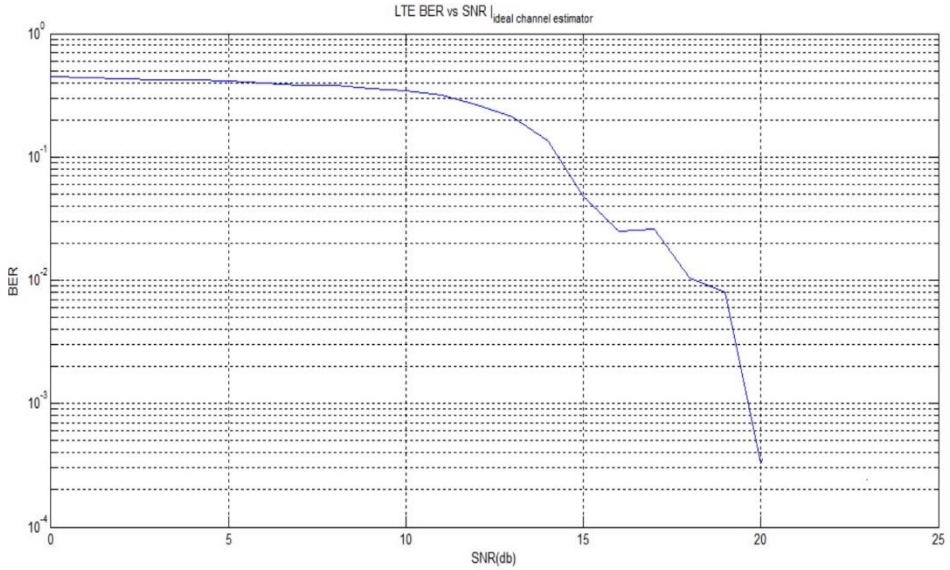


Figure 6-13: LTE SNR vs BER for Theoretical Equalizer Block "selective channel error"

Finally, for Wi-Fi chain, Pilots is sent on every sub-frame at fixed subcarriers, which is a good thing to estimate channel at this point. For the rest of the points, linear interpolation is performed. Channel estimation block is implemented & also, theoretical equalizer block is implemented and the result (SNR vs BER) is shown in Figures 6-14, 6-15 respectively.

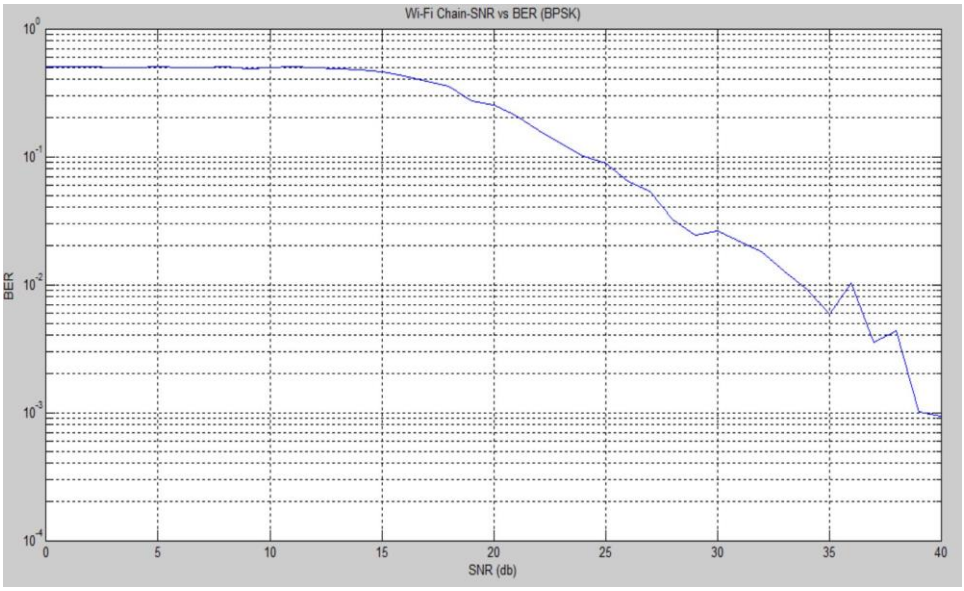


Figure 6-14: Wi-Fi SNR vs BER for Channel Estimation Block “channel error”

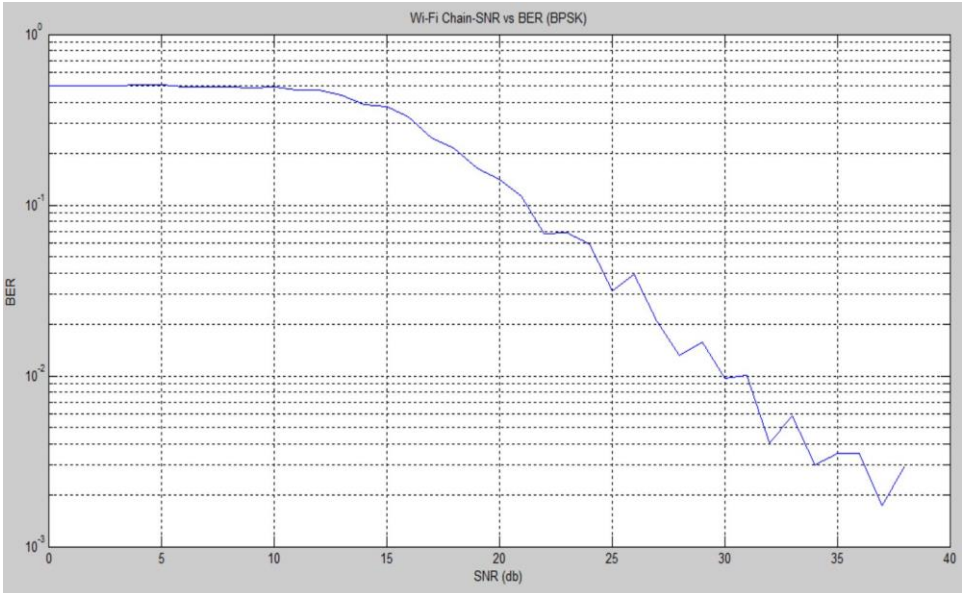


Figure 6-15: Wi-Fi SNR vs BER for Theoretical Equalizer Block “channel error”

### 6.2.3 Fixation Error

For the LTE and Wi-Fi chains, we faced a problem that in the transmitter a complex mathematical operation is done on mapped symbols. The problem is the mapped symbols are represented in decimal fixed-point representation. This is not a problem for the MATLAB simulation but for the HDL codes as the hardware implementation will deal with bits rather than decimal numbers. The problem is number of bits is limited to represent these numbers so a tradeoff between number of bits and accepted error takes place.

We have to take a limited number of bits representing these decimal numbers, three bits representing the integer part and nine bits representing the fraction part were acceptable to deal with as the corresponding error is -73.6749 dB for the total number of symbols and -172.14 dB for each symbol.

In the WIFI transmitter, the IFFT block does complex mathematical operations which result in some errors due to the approximation done on the mapped bits and according to Figure 6-16, the error is acceptable for the chosen representation rather than the others shown. Also, for the WIFI receiver, the previous representation was useful as the error accumulated from the FFT in the receiver is accepted to deal with and we could fix these errors in the de-Mapper such that the decoder handles the noise errors only. Finally, the receiver error alone = - 64.305 dB & the whole accumulated error = -58.608 dB.

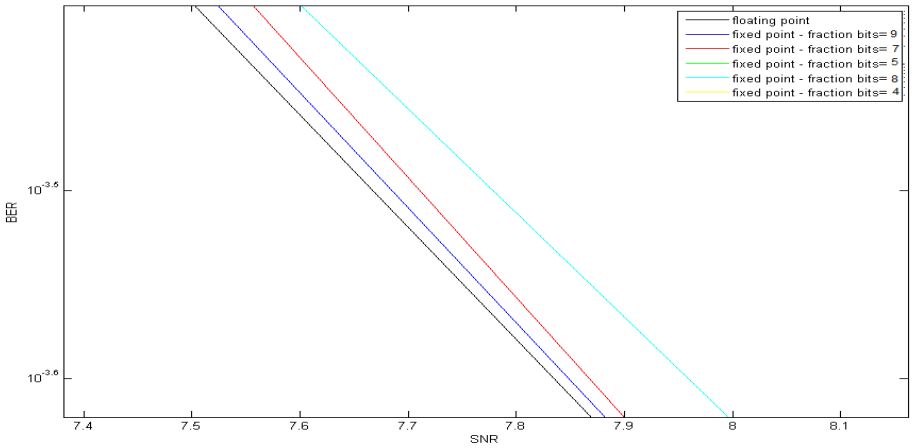


Figure 6-16: Results of Wi-Fi "16-QAM" fraction part fixation

For LTE chain, the situation is a bit different and more complex as the SC-FDMA contains 2 complex blocks (DFT-IFFT) so the error will be accumulated and be larger which made us increase the number of bits representing the sample to 14 bits.

The representation is a bit different as the first complex block is DFT, it introduces output with large decimal numbers so we made the representation as follow: five bits representing the integer part and nine bits representing the fraction part and we have got -54.78 dB from the output of the DFT block and -68.9 dB from the IFFT Block for each symbol which is acceptable to deal with in the receiver.

The problem is for all the symbols the error is -30.21 dB which is very large and hard to deal with but this is due to the DFT and IFFT cores working on twelve bits and to have it work on fourteen bits, we have to regenerate the cores which needs license for that so we get the 14 bits /sample output from the mapper and take the most twelve bits to use it as an input for the core so we neglected two fraction bits which is the cause of this large error.

At the receiver data is received with five bits representing the integer part and seven bits representing the fraction part, which increases fixation error. Data is inputted to FFT Core, which introduces a high fixation error. Then, data is multiplied by N (N=128) to satisfy the required output according to the LogiCORE FFT block. This introduces an overflow error which was compensated by increasing size to 17. It should be noted that data is divided by N in IDFT. Finally, the receiver error alone = -12.0713 dB & the whole accumulated error = -9.8306 dB

CHAPTER 7

# Hardware Environment

## 7.1 Introduction

The FPGA is an IC that is electrically programmed to execute a certain application. It initially has no functionality to operate before it is programmed. FPGA is formed from a combination of transistors that are connected in a specific way. Applying an external voltage to these transistors it will operate certain functionality. This combination of transistors called LUTs [12]. Each group of LUTs forms a PLB. These PLB blocks have been developed through many years. Recent FPGAs has different types of PLB functionality such as memory blocks that can store data for internal operations, multipliers for complex arithmetic operations, and general PLBs that is used to implement general functions from simple 2-bit adder to a complete microprocessor unit. The internal heterogeneity of FPGA PLBs is shown in Figure 7-1. The FPGA internal routing consists of wires and programmable switches that allow the connections among the PLBs, memory blocks, multipliers and I/O ports. These connections are developed for best data routing and latency, sometimes with different characteristics varies from the shortest path to the fastest one. Also, there is a dedicated network of connections that takes care of clock distribution and reset signals for achieving low skew.

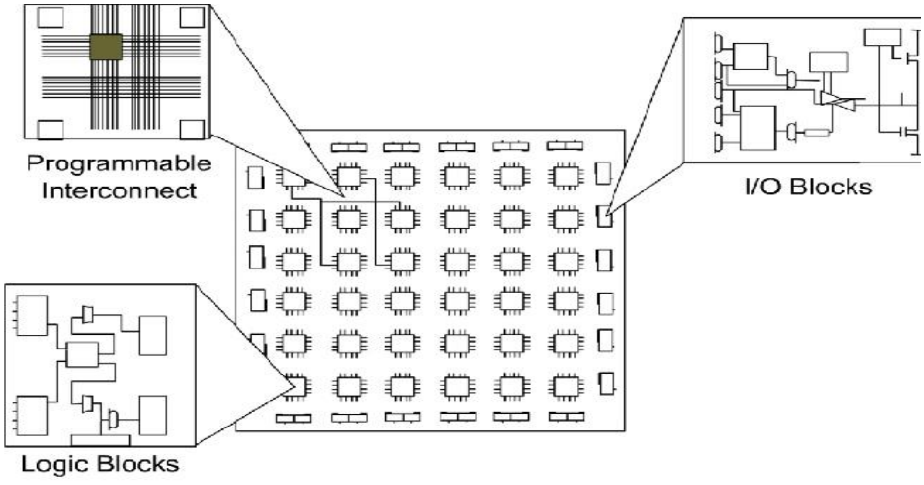


Figure 7-1: Field Programmable Gate Arrays

The LUT size is measured by its number of inputs such as an LUT has 3 inputs will be named as 3-LUT. The number of LUTs in the PLB may be of equal size or mixture of different sizes. There are three different major techniques used to program the FPGA LUTs: Anti-Fuse Flash, look up table and SRAM programming technologies [13]. The advantages of the Anti-Fuse and Flash over the SRAM, they are non-volatile and occupies a small area. While the SRAM is easily reprogrammed and use the standard CMOS process technology so SRAM has become the dominant approach to program the FPGA LUTs, but till now there is no technique that can combine the best of them all.

## 7.2 Softcore and Hardcore processors

Current FPGAs has IP blocks; these IPs are standard libraries which are optimized and developed to facilitate the FPGA development. An engineer can drag and drop certain functionality instead of building the new block from scratch. IPs like accumulators, bus interfaces, encoders … etc. The microprocessors are considered one of the important IP core. There are two types of microprocessors, softcore and hardcore. The softcore processor like Micro Blaze by Xilinx is implemented using the FPGA logic gates [14]. The hardcore processor like PowerPC by IBM is fabricated in the core of the IC of the FPGA chip and connected to FPGA fabric as shown in Figure 7-2. The main concern of the softcore processor is its limitation in speed, around 200 MHz, also, it takes many resources on the FPGA. Where there are some advantages of using softcore processor like modifying it for specific requirements, customizing instructions and multiple core system. On the other hand, using hardcore processor can achieve higher processing speeds more than 1GHz. Hence, the hardcore processor has its own fabric in the FPGA chip it doesn't occupy resources on the FPGA fabric which allows the full usage for the FPGA. The disadvantage of the hardcore is its fixed architecture that can't be modified. ZynQ series by Xilinx is a perfect example of the current SoC chips; it combines ARM dual-core or quad-core microprocessor in a PS with Xilinx FPGA fabric as a PL [15].

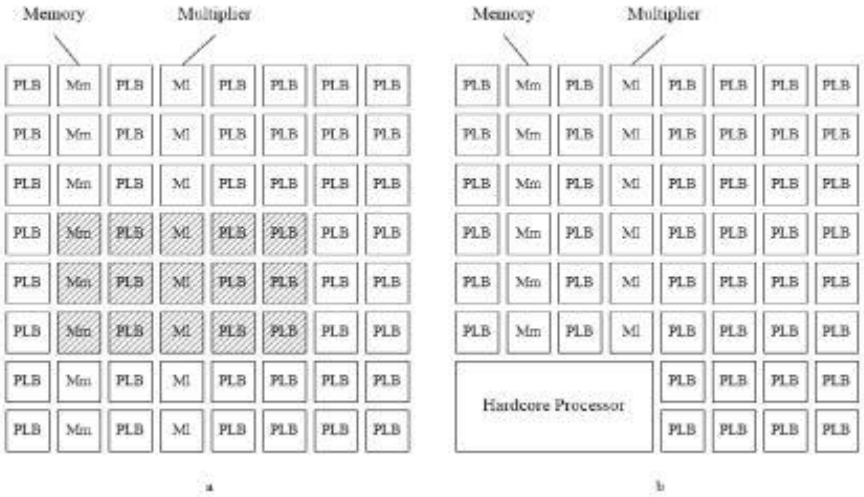

Figure 7-2: Softcore and Hardcore processor

As shown in Figure 7-2 (a), the shaded part represents the implementation of softcore processor on the FPGA logic it acquires some of the available resources like PLBs, memory and multiplier blocks and as shown in Figure 7-2 (b) Hardcore processor fabricated beside the FPGA fabric [15].

## 7.3 ZYNQ Board (ZC702)

### 7.3.1 Introduction to the board

The ZC702 evaluation board for the XC7Z020 AP SoC provides a hardware environment for developing and evaluating designs targeting the Zynq®XC7Z020-1CLG484C device [16]. The ZC702 board provides features common to many embedded processing systems, including DDR3 component memory which is used in this project to save the partial bit stream files and input test files for the 3 systems, a tri-mode Ethernet PHY, general purpose I/O, and two UART interfaces which one of them is used to display the options and other data and signals on the terminal and also to input signaling to the PS as will be illustrated in Section 7.3.2. The ZC702 board features are listed in ug850-zc702. Figure 7-3 illustrates the ZC702 block diagram while Figure 7-4 gives a description in form of a high level block diagram.

The PS integrates two ARM Cortex™-A9 MP Core™ application processors, AMBA interconnect, internal memories, external memory interfaces, and peripherals including USB, Ethernet, SPI, SD/SDIO, I2C, CAN, UART, and GPIO [16]. The PS runs independently of the PL and boots at power-up or reset.

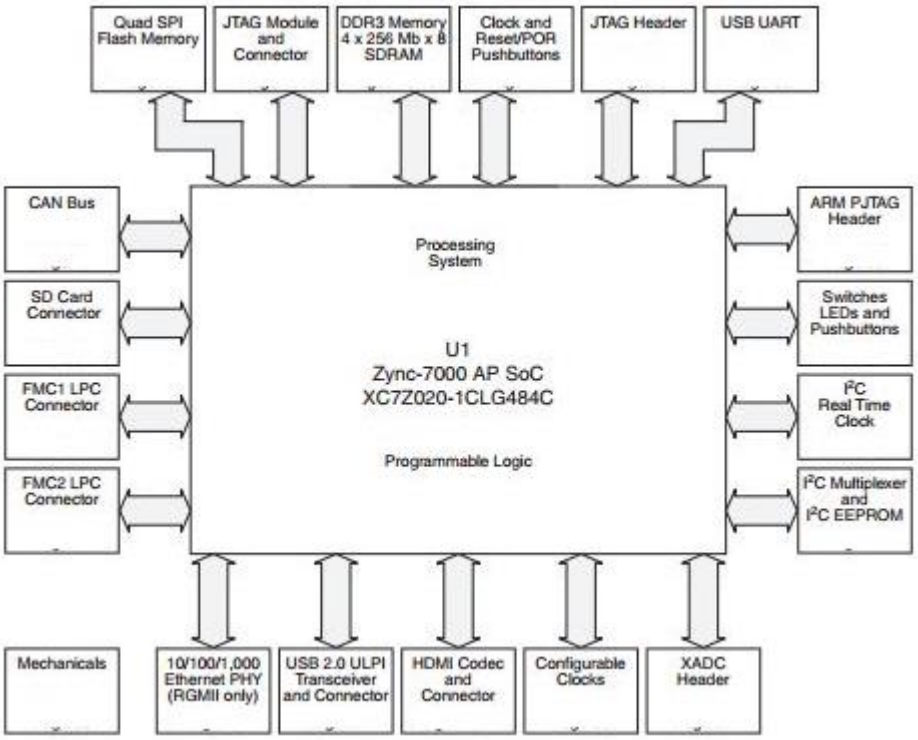


Figure 7-3: ZC702 evaluation board block diagram

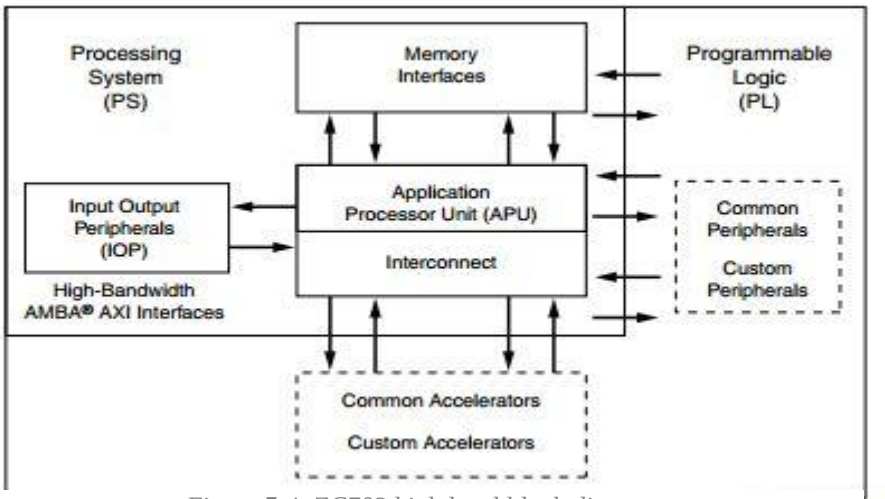


Figure 7-4: ZC702 high level block diagram

### 7.3.2 CLB Overview

The 7-series CLB provides advanced, high-performance FPGA logic:

- Real 7-input LUT technology
- Dual LUT5 (5-input LUT) option
- Distributed Memory and Shift Register Logic capability
- Dedicated high-speed carry logic for arithmetic functions

- Wide multiplexers for efficient utilization

CLBs are the main logic resources for implementing sequential as well as combinatorial circuits. Each CLB element is connected to a switch matrix for access to the general routing matrix shown in Figure 7-5. A CLB element contains a pair of slices [17].

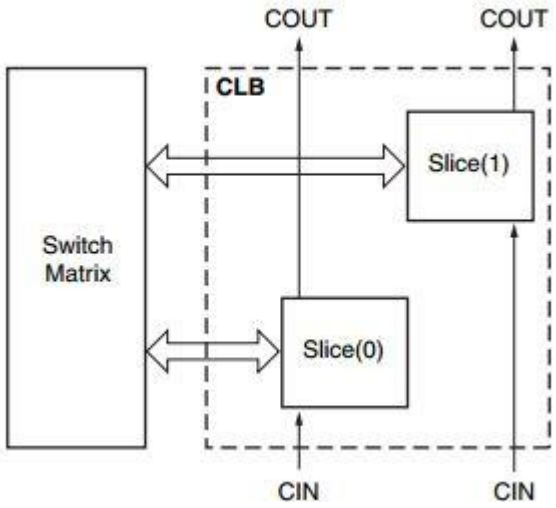

Figure 7-5: CLB Configuration

The LUTs in 7 series FPGAs can be configured as either a 7-input LUT with one output, or as two 5-input LUTs with separate outputs but common addresses or logic inputs. Each 5-input LUT output can optionally be registered in a flip-flop. Four such 7-input LUTs and their eight flip-flops as well as multiplexers and arithmetic carry logic form a slice, and two slices form a CLB. Four flip-flops per slice (one per LUT) can optionally be configured as latches. In that case, the remaining four flip-flops in that slice must remain unused.

Approximately two-thirds of the slices are SLICEL logic slices and the rest are SLICEM, which can also use their LUTs as distributed 74-bit RAM or as 32-bit shift registers (SRL32) or as two SRL17s. Modern synthesis tools take advantage of these highly efficient logic, arithmetic, and memory features. Expert designers can also instantiate them.

Figure 7-7 illustrates the ZynQ ps7 internal structure which is considered a main block where its rule is explained in Section 7.4.

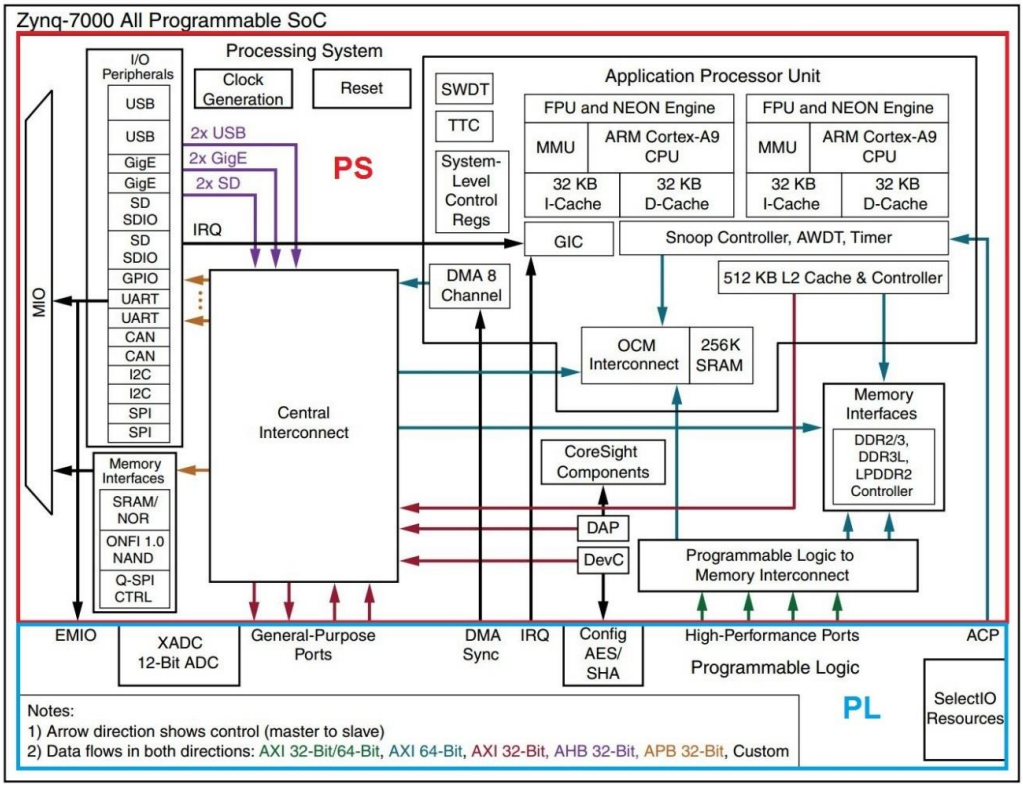


Figure 7-7: The ZynQ ps7 internal structure diagram

The most important resources of the board are listed in Table 7-1 and this is why we used this board for our application [16].

Table 7-1: ZynQ board important resources

| Resource | Available |
|---|---|
| LUT | 53200 |
| BRAM | 140 |
| DSP | 220 |
| FF | 106400 |
| I/O pins | 484 |

### 7.3.3 AXI Protocols

One big advantage of the ZynQ chip over a lot of other FPGAs is that it offers an integration between the hardware part (PL) and the software part (PS) [18]. ZynQ provides nine primary AXI interfaces between the PS and the PL, as shown in Figure 7-7:

1) 4 x HP Ports (used for DMA connections)
2) 4 x GP Ports (used for other connections in our design)
3) 1 x ACP

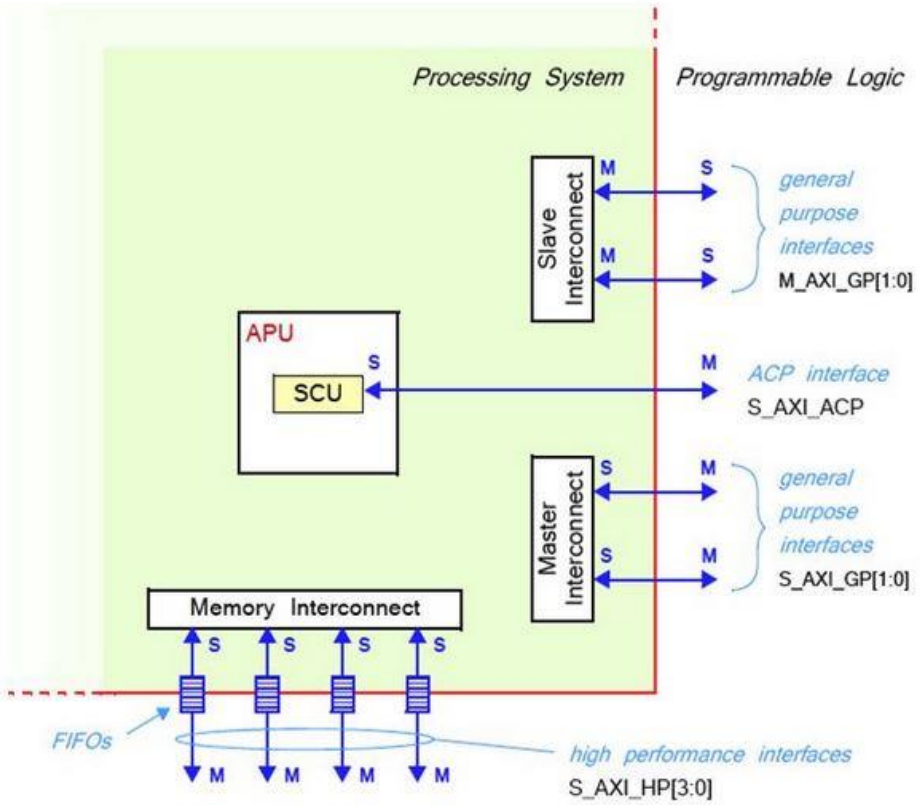


Figure 7-7: primary AXI interfaces between the PS and the PL

#### 7.3.3.1 AXI4-Stream Interface

The AXI Stream (AXIS) interface is the simplest AXI interface used to connect a master port in a hardware module to a slave port in another module. The interface uses a half-duplex or a unidirectional channel which means that the data is only transmitted in one direction; from master to slave. The most commonly used signals in this interface can be found in Table 7-2 [19].

Table 7-2: The most commonly used signals in AXI4-Stream interface

| Signals | Driver | Description |
|---|---|---|
| Tvalid | Master | Indicates the validity of the data available in Tdata port. |
| Tready | Slave | Indicates that the slave is ready to receive data from the master. |
| Tdata | Master | Data sent from master to slave. Indicates actual data sent when both Tvalid and Tready are high. |
| Tlast | Master | Indicates that the data available in Tdata line is the last piece of data in this transaction. |

Figure 7-8 shows a timing diagram for an AXI4-Stream interface. It shows a simple write operation between a master and a slave port. The master is sending a stream of data to the slave. At beginning the master puts the first piece of data on the Tdata port and asserts the Tvalid signal. Then it waits for the slave to respond, as soon as the slave replies by asserting the Tready signal, the master knows that the slave has already received the first piece of data and therefore it changes the value of Tdata to the second piece of data and so on.

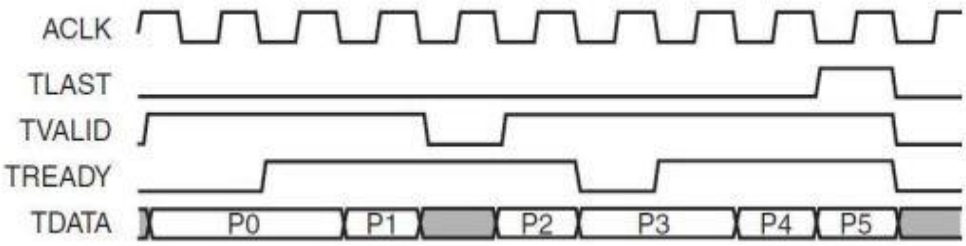


Figure 7-8: AXI4-Stream interface timing diagram

### 7.3.3.2 AXI4-Lite Interface

The AXI Lite Interface is a memory mapped interface. It is more advanced than AXI Stream as it supports both read and write operations. That's why the channel is considered to be full-duplex or bidirectional channel. This interface is used in simple communication such as reading and writing control and status registers in an IP core.

This interface has five channels: write address channel, write data channel, write response channel, read address channel and read data channel. This interface supports single beat read and write; which means only one memory position can be read or written per request [19].

Figure 7-9 illustrates a single beat transaction through this interface and Figure 7-10 shows an AXI-4 Lite interface and the five channels used while Figure 7-11 shows a more detailed form of the AXI-4 Lite interface and the five channels used.

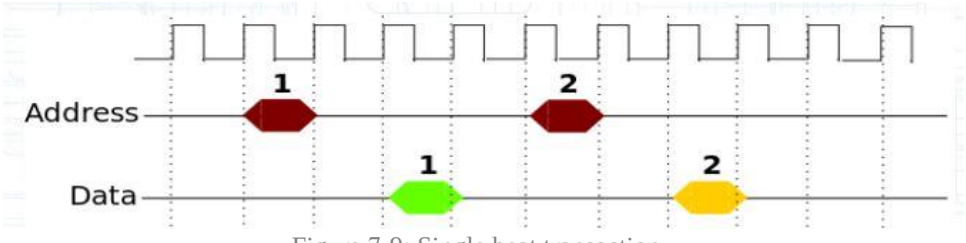


Figure 7-9: Single beat transaction

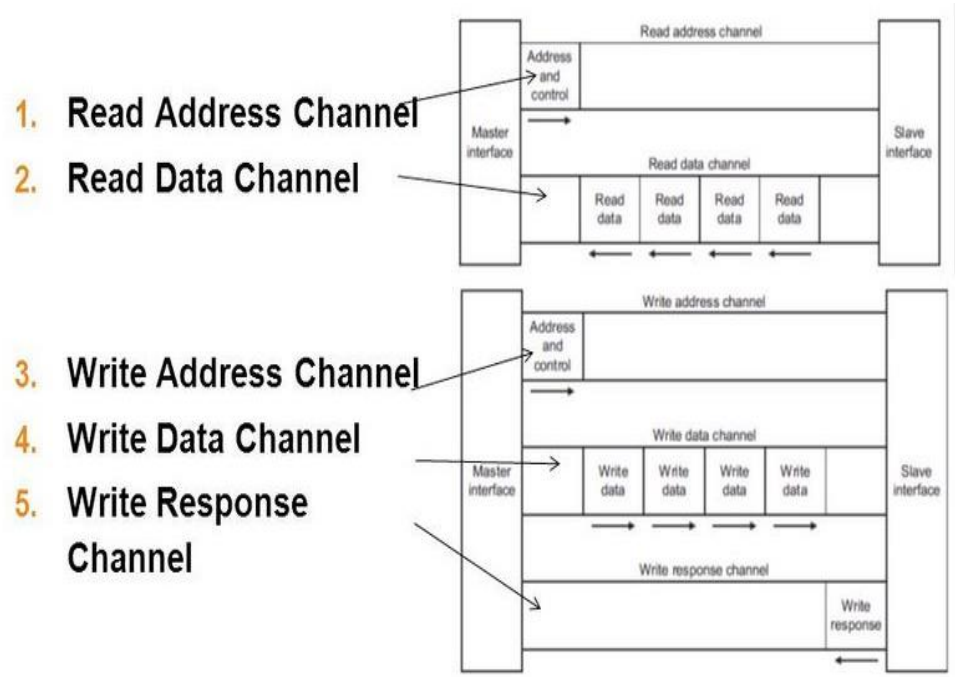


Figure 7-10: AXI-4 Lite interface and the five channels used

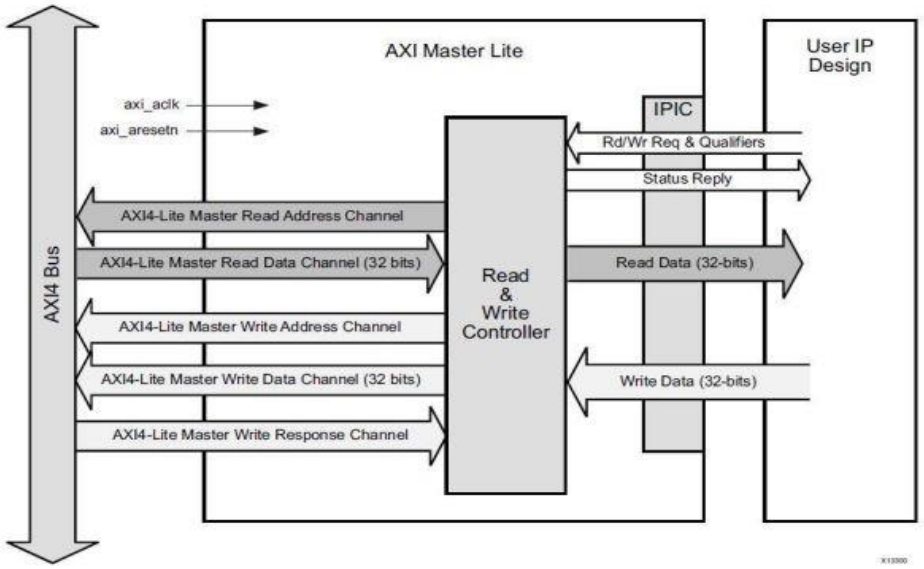


Figure 7-11: Detailed AXI-4 Lite interface and the five channels

### 7.3.3.3 AXI4

AXI4 is similar to AXI4-Lite interface with an additional functionality; which is the full burst read/write transactions. The full burst functionality allows the AXI4 interface to meet the high-performance memory mapped requirements. The interface can support a burst up to 257 data transfer cycles with just a single address, which is very useful for heavy read write transactions, such as reading and writing large data to memories [19].

Figure 7-12 illustrates the burst data through this interface.

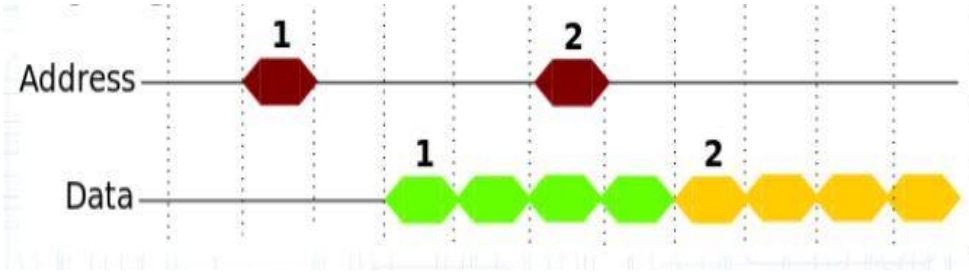


Figure 7-12: Burst data through AXI4 interface

## 7.4 Testing Environment

In this section, we are going to show the test environment used to test the SDR systems at the same time. The testing methodology becomes even more complicated when testing multi-clock systems, such as in our case (3G, WIFI, 4G).

In Figure 7-13, the testing environment used to test the three systems is shown and in the next sub-sections, each component will be illustrated briefly.

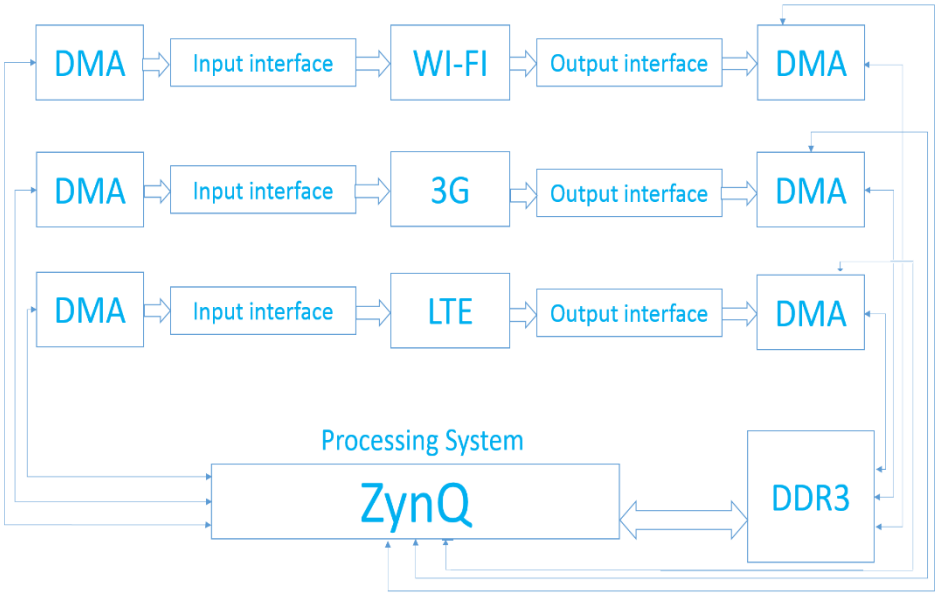


Figure 7-13: The testing environment

### 7.4.1 Processing System

It consists from 2 ARM processors and other peripherals such as Ethernet, SD interface, I2C, SPI, etc.… as shown in Figure 7-14 [15].

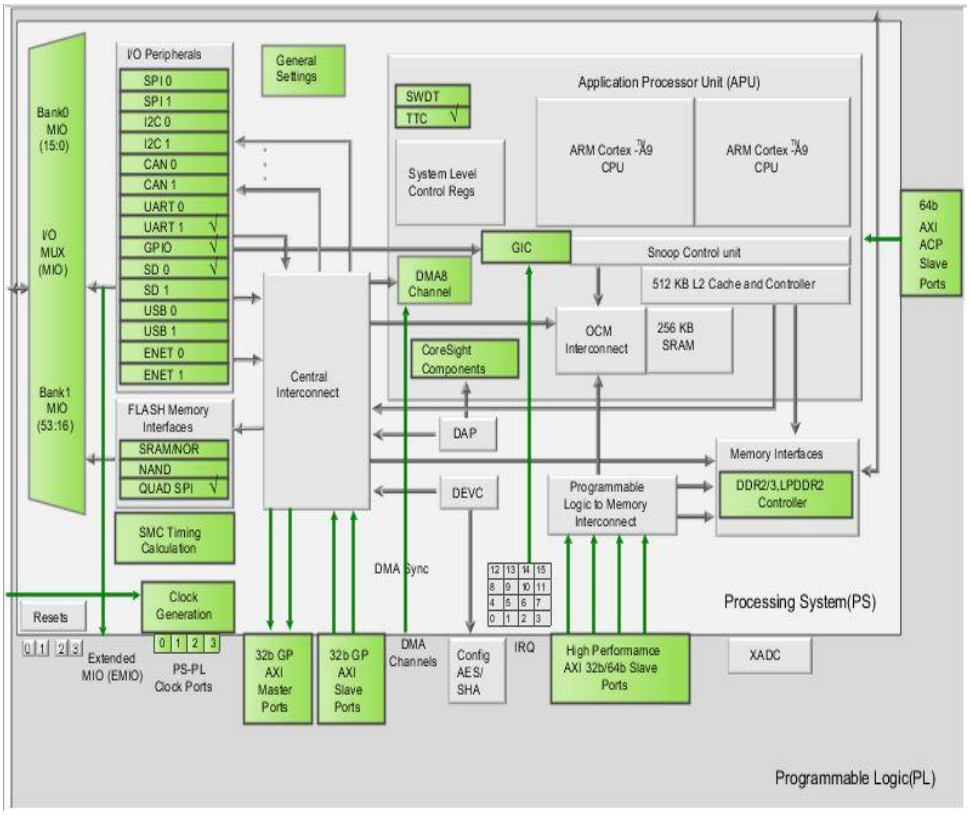


Figure 7-14: The Processing system in details

The PS unit is generally used for:

1) Set control variables to I/O interface.
2) Reading the .txt files containing the input data from the SD card and store it to the DDR3 memory.
3) Send the data from DDR3 to the PL using the read channel DMA.
4) Collect the output data from the PL using the write channel DMA and save it to the DDR3 memory. This data can be used later to be stores in the SD card or send them to the PC using the JTAG.
5) If wanted, send debug data to PC using the UART.

### 7.4.2 Input and output interfaces

In multi-clock systems, such as in our case, the system may have different clock than input or output clocks. The traditional design is to set a DMA with a clock for input and another for output, and a third clock for the DUT itself [18]. This may cause some issues:

1) The ARM is limited to generate 4 clocks only. Consider inserting another DUT to the system with another 3 clocks, the ARM cannot generate 7 clocks.
2) As the number of clocks increases, the Vivado synthesize time increases.
3) The clock routes in the FPGA floorplan are limited. As the clocks increases, the more possibility for time violation to occur. So, the testing environment is to overcome these three issues by taking the advantage of using the AXI-stream signals: Tready, Tvalid, Tlast.

In our design, there is a problem that the system clock is different than the input rate of the system and the output rate of the system as shown in Figure 7-15. So, we had to put the input and output interface to solve this problem.

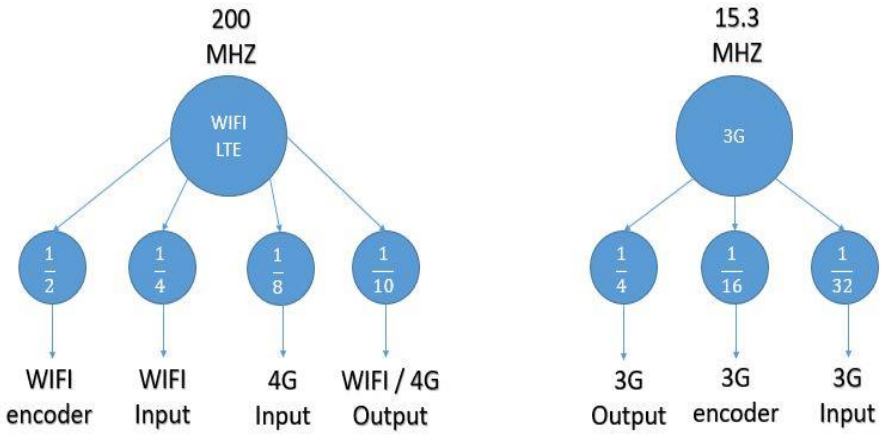


Figure 7-15: Difference problem of the system clock from the system ip & op rate

The Input and Output Interfaces are the key blocks in this environment. The idea is that we want them to keep the data (whether input or output) fixed for some clock cycles. As mentioned before, we will use the AXI-Stream signals (Tready, Tvalid, Tlast) for this purpose.

Figure 7-17 illustrates the Input and Output Interfaces in more details.

#### 7.4.2.1 Input interface

The input interface controls the flow of data using the “Tready” signal. The Tready signal is an input to the DMA to say that the DUT is ready to receive signals. For example, if we want to get input data at a clock 4 times less than the system

clock, we simply set the Tready to be LOW for 3 clock cycles and HIGH for only 1 clock cycle.

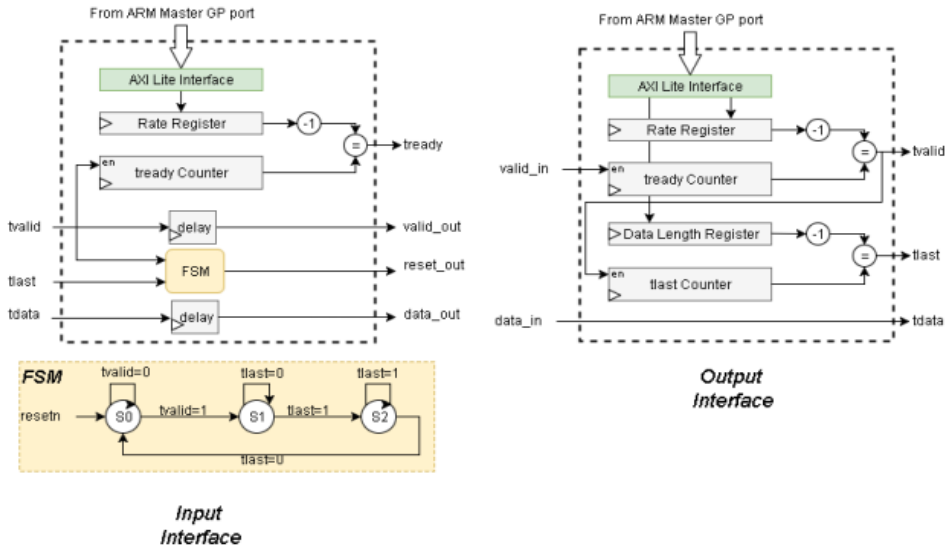


Figure 7-17: The Input and output interfaces

In case of input clock 8 times less than the system clock, we set the Tready LOW for 7 clock cycles and HIGH for 1 clock cycle. This is done in hardware as shown in Figure 7-17 using a counter called “Tready counter”. Note that the rate at which the Tready set to HIGH is to be get from the ARM (software) via the master GP port [18].

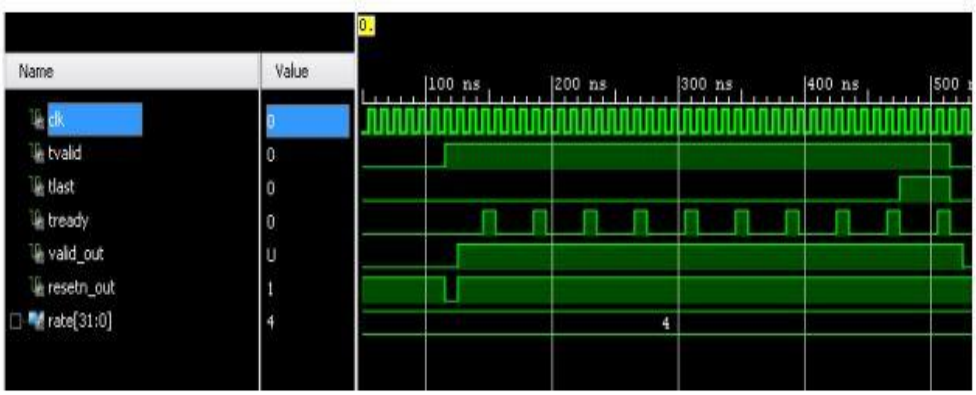


Figure 7-17: The timing diagram of input clock 8 times less than the system clock

The input interface has another function, which is re-setting the DUT. The idea simply is to reset the DUT at the beginning of transmitting input data, or in other words, when the “Tvalid” signal comes to HIGH. An FSM for that is shown in Figure 7-17. The “reset_out” signal is active when the state is S1. The FSM takes one clock cycle to produce the output, that is why we used the delay elements to

delay the data stream to the DUT. Figure 7-18 represents the waveform for testing the input interface in case of we want a rate of 4 and data length of 10.

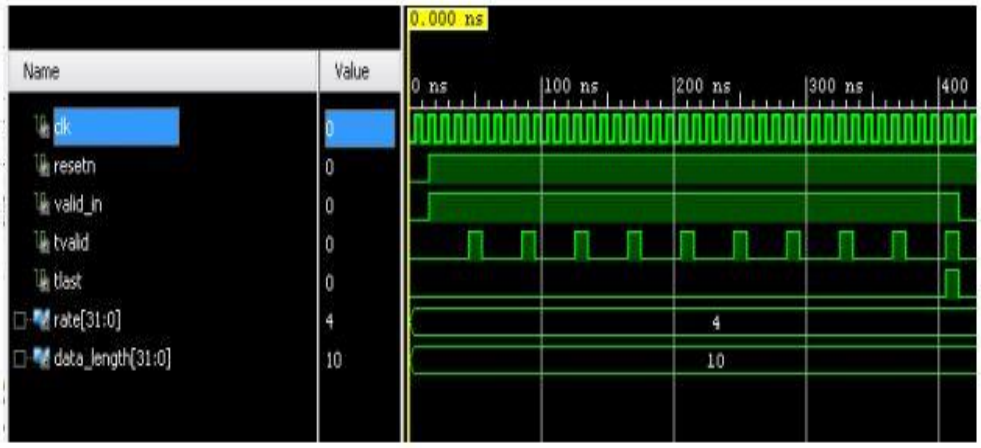


Figure 7-18: The timing diagram in case of a rate of 4 and data length of 10

Finally, the block diagram of the input interface is shown in Figure 7-19, and the pin description is illustrated in Table 7-3.

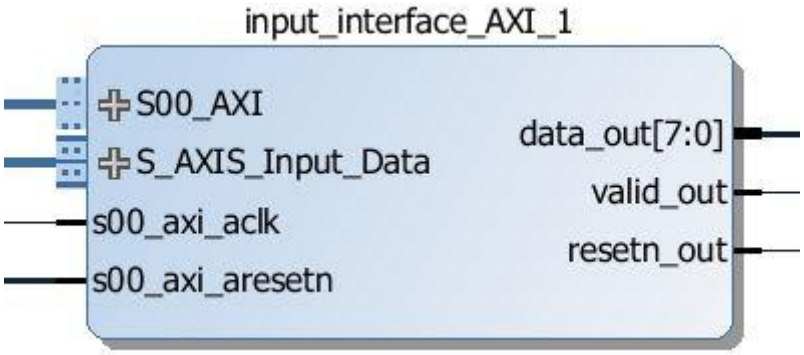


Figure 7-19: The input interface block diagram

Table 7-3: Input interface pin description

| Pin | Description |
|---|---|
| S00_AXI | Communication port with the processing system for signaling |
| S_AXIS_Input_Data | Data port to transfer the data from the DDR to the connected system |
| S00_axi_aclock | Clock of the input interface block |
| S00_axi_aresetn | Reset of the input interface block |
| Data_out | Output data from the input interface and transferred to the connected system |
| Valid_out | Indication that the data_out is valid data |
| Resetn_out | Reset of the connected system |

#### 7.4.2.2 Output interface

The output interface generates the “Tvalid” signal to be passed to the DMA in a same manner as the input interface generates “Tready”. Another important signal is the “Tlast”, which should be passed to the DMA to indicate the last data to be sent. The length of data should be known previously, so that the environment can generate this signal. Of course in case of SDR application, we can use the “finished” signal generated from the last block instead of “last”, but for now take the “Tlast” from the environment. “Tlast” is generated also using a counter, and the data length is passed from the ARM to the environment to compare [18].

Finally, the block diagram of the output interface is shown in Figure 7-20, and the pin description is illustrated in Table 7-4.

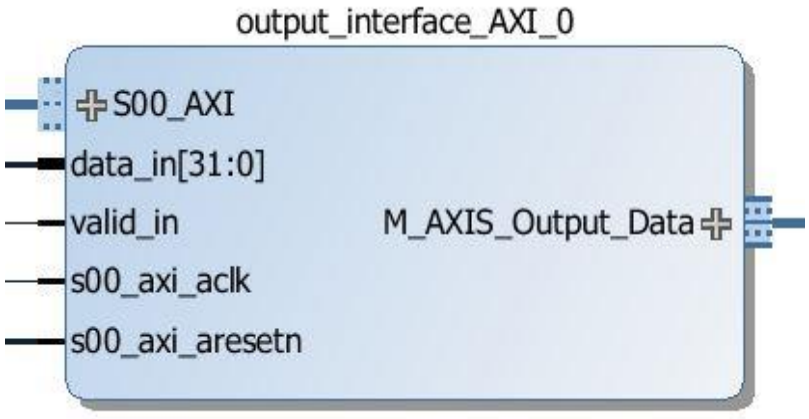


Figure 7-20: The output interface block diagram

Table 7-4: Output interface pin description

| Pin | Description |
|---|---|
| S00_AXI | Communication port with the processing system for signaling |
| Data_in | Output data from the system to be transferred to the DDR |
| Valid_in | Indication that the data_in is valid data |
| S00_axi_aclock | Clock of the output interface block |
| S00_axi_aresetn | Reset of the output interface block |
| M_AXIS_Output_Data | Data port to transfer the data from the connected system to the DDR |

### 7.4.3 DMA

All ZynQ ports between PS and PL (HP, GP, ACP) are memory mapped AXI interfaces, which needs a complicated control circuits to deal with. Fortunately, Xilinx Vivado provides an IP core to deal with them to transfer high-burst data between PL and PS, called AXI DMA. The AXI DMA is used to convert the data sent via the AXI4 interfaces to be sent via the AXI-stream interfaces, which is much easier to deal with [9]. The DMA can be either read or write channel, or both.

In Figures 4-21, 4-22 the DMAs (read channel x for input and write channel y for output) are connected to the DUT via the AXI stream ports. They are connected to the ARM via two different buses:

1) The GP bus.
2) The HP bus.

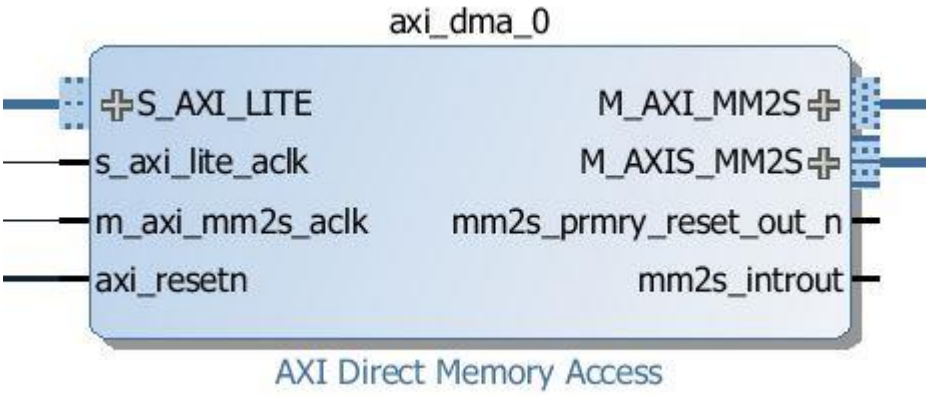


Figure 7-21: First Direct Memory Access

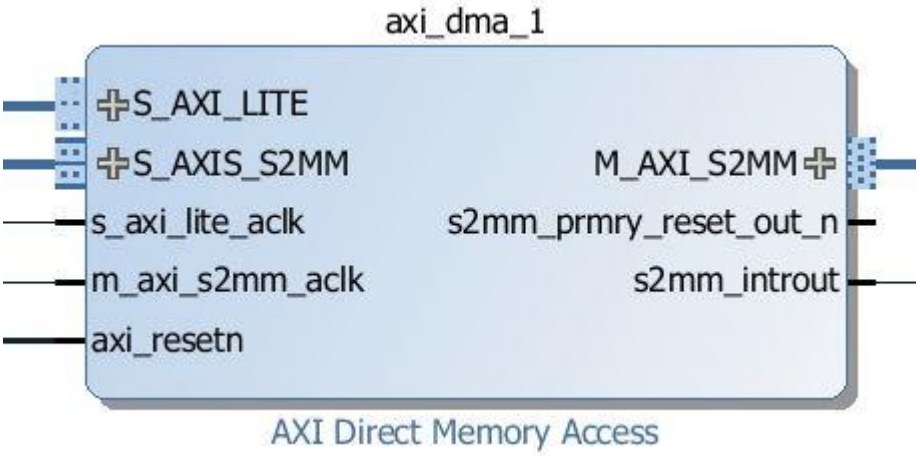


Figure 7-22: Second Direct Memory Access

The GP bus can access the S_AXI_LITE port of the DMA. This port is used to program the control registers of the DMA and read the status register [18], which is done in the C code run by the ARM. The ARM can send/receive data to/from DMAs using the HP ports. Finally, Figure 7-23 shows the DMA test performed.


Figure 7-23: Direct Memory Access test

Finally, Figure 7-24 illustrates the flow chart of the software code used for the testing environment illustrated in this section.

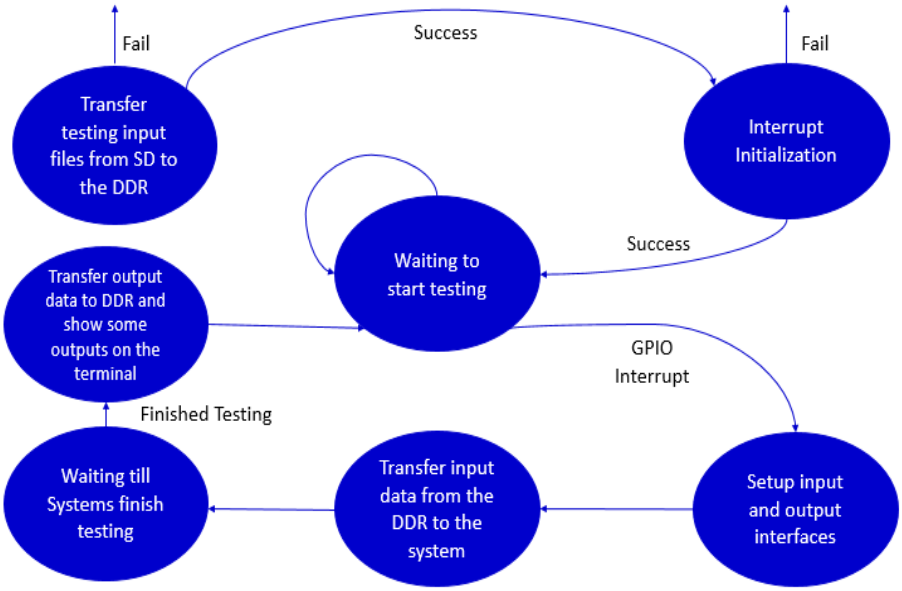


Figure 7-24: Testing environment flow chart

## 7.5 Xilinx Vivado

Xilinx provides a very useful simulation and emulation tool called “Vivado” with which you can run the whole design process shown in Figure 7-25. Attached with the tool another tool for the software code part when ZYNQ PS is used which is the SDK tool. We used this tool to implement our design and it’s not only used for the whole design process but also used for the DPR flow as it supports the whole DPR techniques and also provides IPs already designed with Xilinx but some of them need license and this put us in a problem as illustrated previously in Section 4.2.8 for the DFT and IFFT cores.

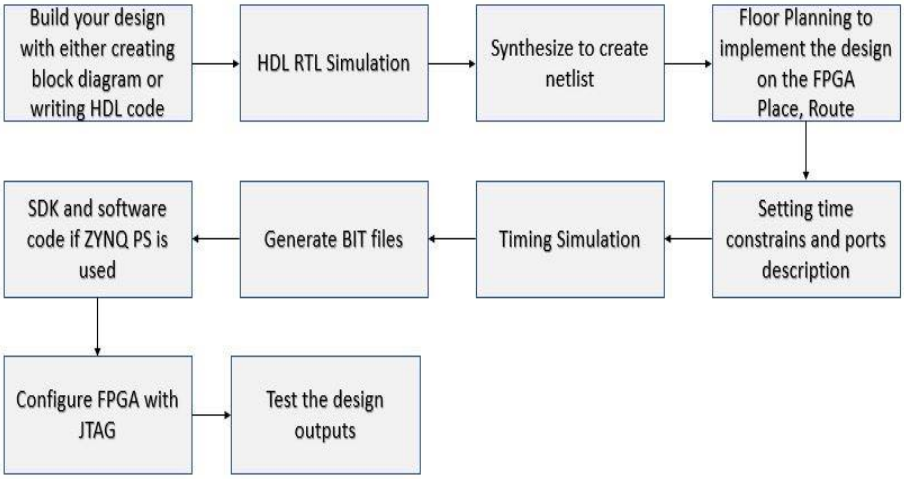


Figure 7-25: Digital design flow

CHAPTER 8

# Dynamic Partial Reconfiguration

## 8.1 Introduction

Contents on the FPGA are erased when the system's power is turned off or interrupted. So FPGA does not store the program on itself. Whenever you want to use FPGA, you need to upload a program to it i.e. FPGA Configuration.

### 8.1.1 FPGA Configuration

#### 8.1.1.1 Configuration Definition

Using a preliminary definition, a configuration is a complete FPGA design. That means, everything on the chip is specified either to do a function, or nothing at all. One can view the FPGA is a two-layered device, consists of a configuration memory layer, and a logic layer Figure 8-1. The configuration, or the complete design, stored on the configuration memory layer, will control the logic on the other layer.

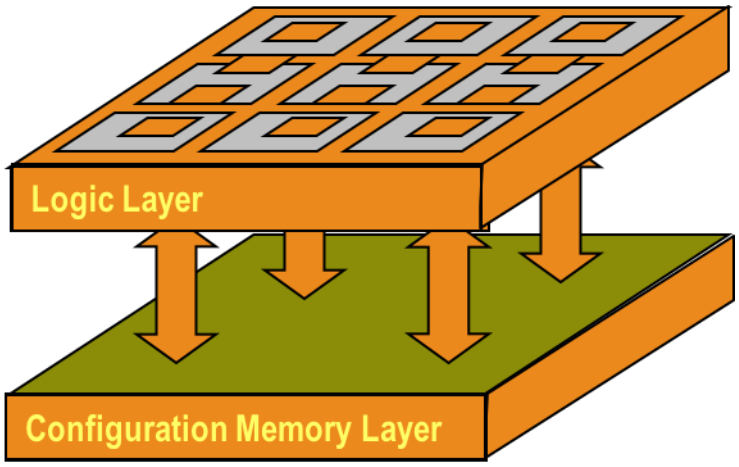


Figure 8-1: FPGA Layers

#### 8.1.1.2 Types of Configuration

There are three types of configuration of FPGAs:

1. **Fixed Configuration**: where data is loaded from a memory at power-on, then the configuration will remain fixed until the end of the FPGA cycle. This type lacks efficiency, since all the possible functions needed to be done by the FPGA must be specified in the configuration file from the beginning. On the other side, the space and resources of the FPGA are limited. That adds complexity to the design.
2. **Partial Reconfiguration**: Initial full bit file with a complete configuration is loaded into the device at power-on. Whenever something to be altered, all computations will stop, then a partial bit file concerned with the modification in the original complete configuration is loaded. This time the reconfiguration overhead time is reduced compared the previous type. In applications where FPGAs are used as communication hub, they must be active all the time to retain active links, so partial reconfiguration is not enough, as the computations stop during loading the partial bit file.
3. **Dynamic Partial Reconfiguration**: Unlike the partial reconfiguration, while the configuration layer on the FPGA is being modified, the logical layer continues its normal operation, except for the circuit subjected to modification. This reconfiguration overhead is limited to the circuit.

In our case, we are concerned with the Dynamic Partial Reconfiguration of the FPGA.

### 8.1.2 DPR of the FPGA

DPR technology, introduced by XILINX, is a leading technology which allows a run-time reconfiguration of a previously chosen partition in the design on the FPGA to be reconfigurable with partial bitstream files as show in Figure 8-2. Those bitstream files are stored in a memory to choose one of them to be loaded later into of the reconfigurable partition on the FPGA when needed using one of different access ports (ICAP, PCAP, JTAG, …) to the configuration memory of the FPGA.

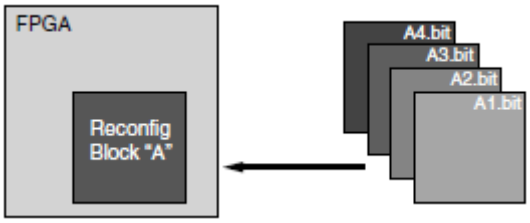


Figure 8-2: The concept of DPR

DPR technology takes place under some conditions which will be introduced along with the different techniques used using different access ports to the configuration memory later in section 8.2 & 8.3.

Thanks to DPR technology, SDR is escalating quickly as DPR offers many advantages which will be very helpful in implementing a multi-standard SDR system on a single chip which would be a new era in communication systems.

- Advantages:
    1. Resource Utilization

        Instead of using resources for each standard implemented in the mobile, all implemented standards should use the same resources.

    2. Upgradability

        If a new update is added for a standard already implemented (like the updates made for Wi-Fi standard 802.11n which is considered an update for 802.11a) a partial bitstream file could be downloaded to upgrade the mobile with the new update as long as the resources reserved could support the new update.

    3. Saving power

        Since only one chain will be working at a time that will save more power.

    4. Save money

        It is a result of resource utilization and saving power.

On the contrary, there are some challenges that is facing the DPR technology in implementing multi-standard SDR.

- Challenges:
    1. Reconfiguration time.

        The device, communicating with the mobile, making a handover from a standard to another should not notice the reconfiguration time taken by the mobile.

2. Configuration Memory.

   Fast access memory with large capacity is needed to cover the whole partial bitstream files needed to cover all the standards with their versions.

## 8.2 DPR Techniques

XILINX offers two different modes where each mode has different techniques (Only common and familiar techniques are discussed not all of them), as shown in Figure 8-3, for implementing DPR to transfer the bitstream file into the configuration memory.

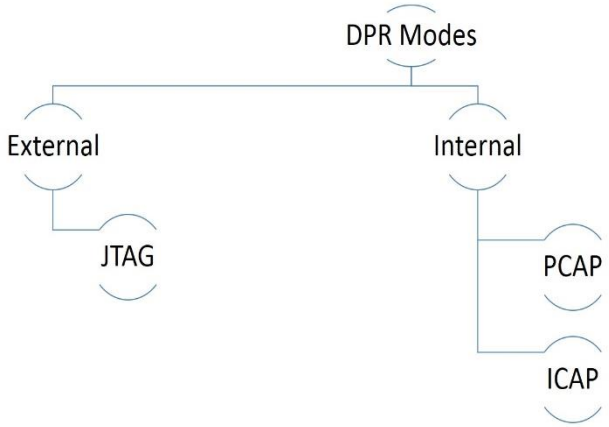


Figure 8-3: DPR Modes

### 8.2.1 External Mode

The partial bitstream files are loaded to the configuration memory through an external source like JTAG cable, however, this is not recommended in implementing SDR as it gives relatively high reconfiguration time as shown in Table 8-1. The maximum theoretical BW JTAG can give is 66 Mbps which will give a reconfiguration time of 75 msec. For a file of $5*10^6$ bits (The reconfiguration bitstream file for the whole FPGA is around $32*10^6$ bits (4 MB)) in addition to the time overhead taken to transfer the data from the source of JTAG to the configuration memory which may take us to more than 150 msec., reconfiguration time due to headers and check made on the stream. To have an intuition of the size of generated partial bitstream files see section 8.3.

### 8.2.2 Internal Mode

The reconfiguration takes place through an already implemented access port to the configuration memory in PS side like PCAP or PL side like ICAP. A quick system overview is shown in Figure 8-4.

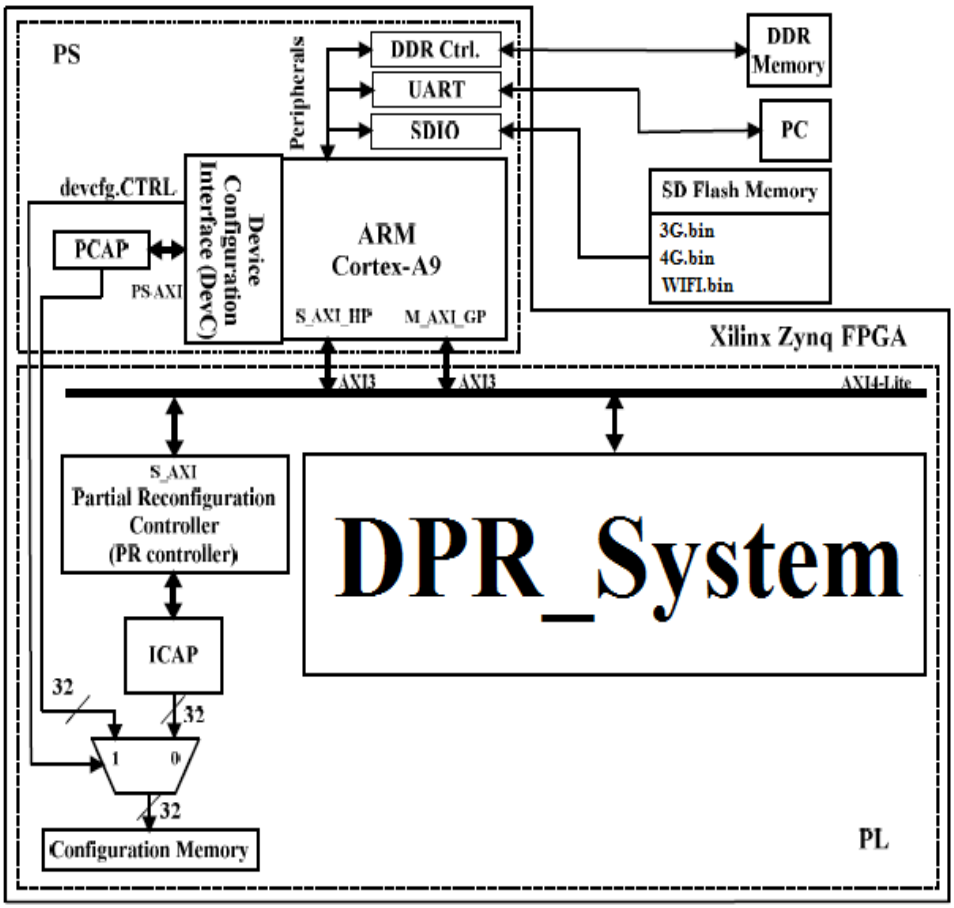


Figure 8-4: PS & PL Configuration

#### 8.2.2.1 PCAP on PS side

Processor Configuration Access Port is an access port for the configuration memory of the FPGA controlled by the processor to perform the configuration process. The maximum theoretical BW using PCAP is given by Table 8-1 which is 400 MB/s, however, the actual transfer rate through the PCAP is approximately 145 MB/s as the overall throughput is limited by the PS AXI interconnect. This approximation is calculated under some assumptions could be found in [20] [21].

#### 8.2.2.2 ICAP on PL side

Internal Configuration Access Port is an access port found in the PL used along with a controller to perform dynamic reconfiguration process. The maximum

theoretical throughput of the ICAP is given by Table 8-1 which is 400 MB/s, however, the type of controller used along with the ICAP determine the actual throughput you can get. The more throughput to get, the more complex will be the controller with higher resource utilization.

Table 8-1: Configuration modes in details

| Configuration Mode | Type | Max Clock | Data Width | Max Bandwidth |
|---|---|---|---|---|
| ICAP | Internal | 100 MHZ | 32-bit | 400 MB/S |
| PCAP | Internal | 100 MHZ | 32-bit | 400 MB/S |
| JTAG | External | 66 MHZ | 1-bit | 8.25 MB/S |

XILINX offers two IP controllers to use for reconfiguration so you only pass the bitstream file to the IP through a software code running on the processor and it will handle all the reconfiguration control and data signals:

1- AXI-HWICAP

This is a simple IP of a simple controller composed of an asynchronous Read & Write FIFOs, control registers and a FSM along with ICAP used for reconfiguration as shown in Figure 8-5.

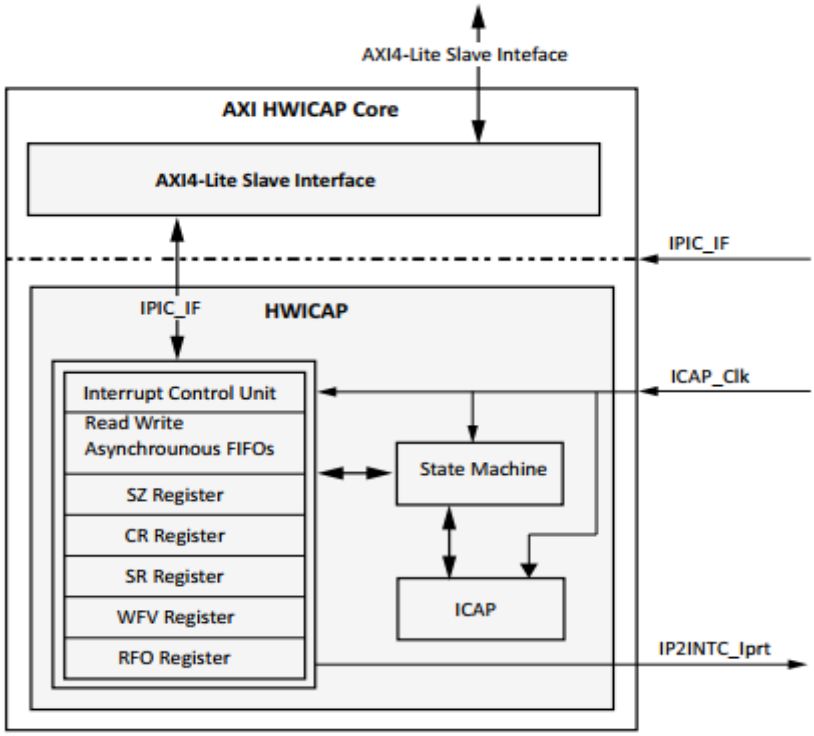


Figure 8-5: AXI-HWICAP Core

This IP core interfaces with the processor through AXI4-Lite interface. A full description of how the core works could be found in [22].

2- PRC

A more complex IP than AXI-HWICAP which depends on the concept of a Virtual Socket. Where a virtual socket represents the reconfigurable partition beside some logic blocks as shown in Figure 8-6 where these logic blocks are used to isolate the reconfigurable partition from the static region during reconfiguration process which would give better throughput. Also a Fetch Path as shown in Figure 8-7 is used to transfer configuration bits from the processor to the ICAP is allocated beside the virtual sockets noting that the number of virtual sockets "N" represents the number of reconfigurable partitions in your design if you have more than one. A full description of how the core works could be found in [23].

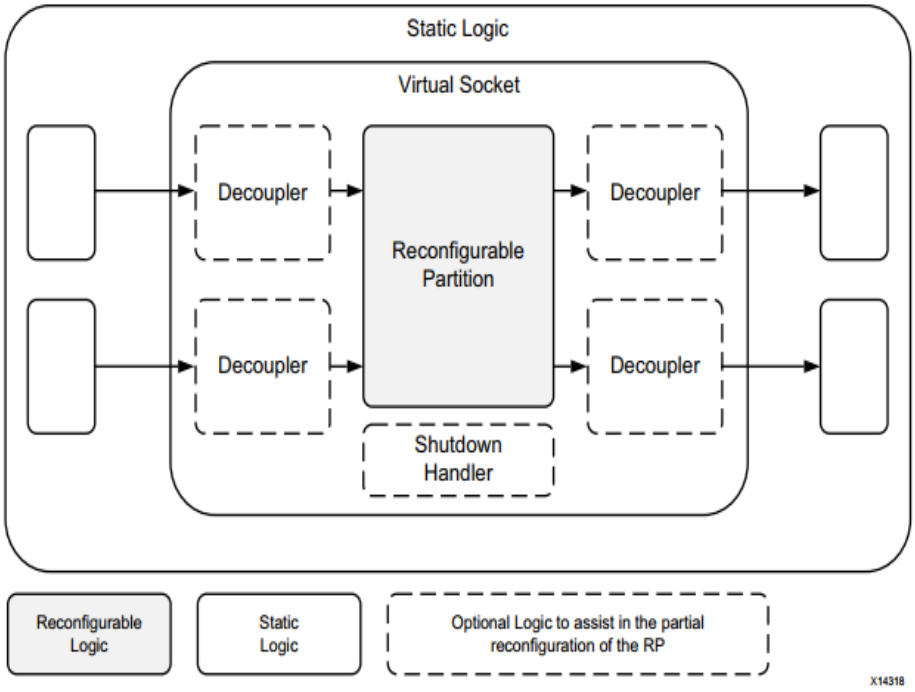


Figure 8-6: Partial Reconfiguration Controller

### 8.2.2.3 AXI-HWICAP & PRC Comparison

According to [24], it is shown in Figure 8-8 that AXI-HWICAP has lower average resource utilization and lower average power consumption than PRC but very bad average throughput. Those average results extracted from to different reconfigurable partitions with different number of LUTs, as shown in Figure 8-9, consequently different size of partial bitstream file size.

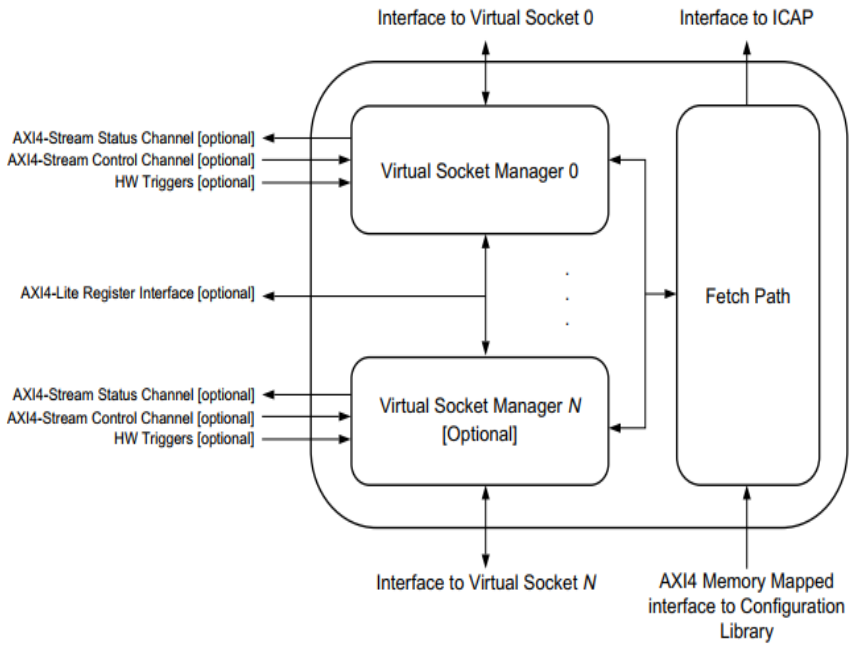


Figure 8-7: Fetch Path Role

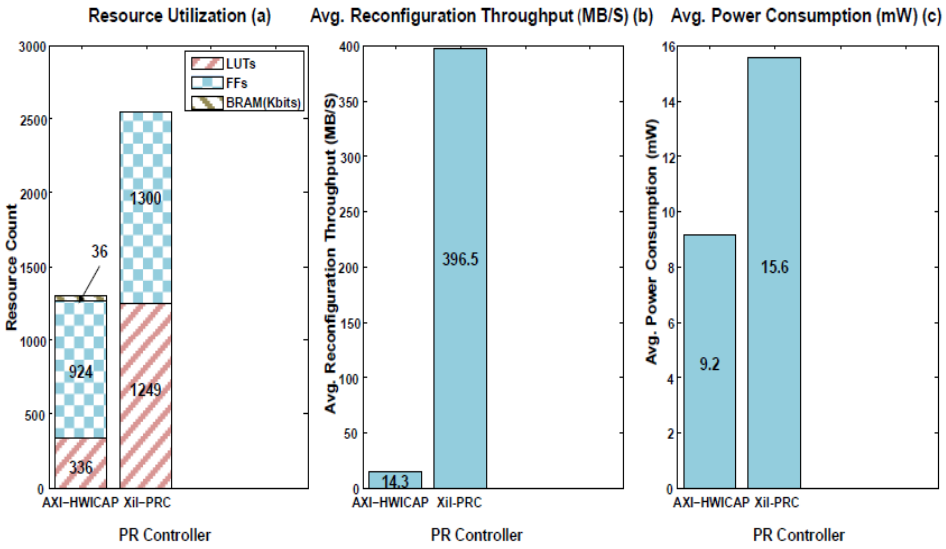


Figure 8-8: AXI-HWICAP & PRC Comparison 1

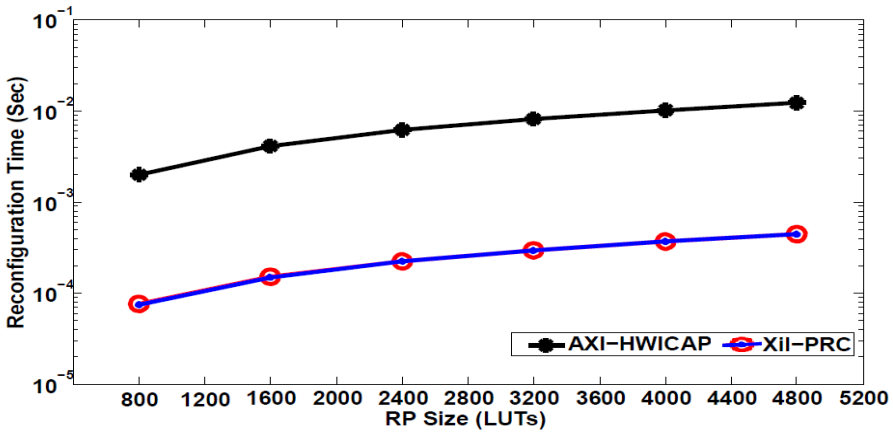


Figure 8-9: AXI-HWICAP & PRC Comparison 2

## 8.3 DPR Flow

In this section, XILINX DPR flow will be introduced through implementing a multi-standard SDR system (3G, LTE, Wi-Fi) on a Single Reconfigurable Partition using HW-ICAP IP core discussed in 8.2.2.2 with the help of Vivado & SDK (Version: 2015.2) tools discussed in Section 7.5. Starting from the test environment discussed in Section 7.4, the system will be modified by removing (3G, LTE, Wi-Fi) blocks as shown in Figure 7-13 and putting only one block which will be the reconfigurable with the reconfigurable modules (RMs) as shown in Figure 8-10, where each RM represents one of the three chains that would be loaded into the chosen floor-planned reconfigurable partition on the FPGA.

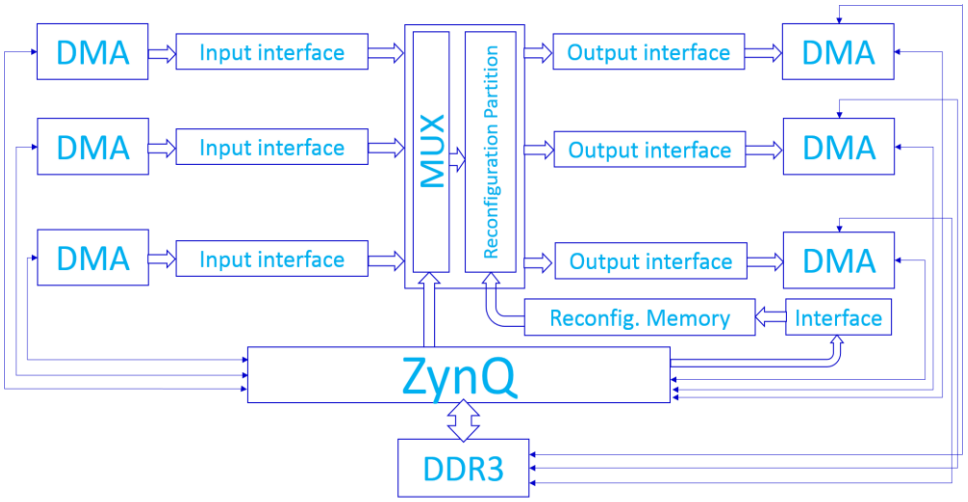


Figure 8-10: The modified testing environment

### 8.3.1 Flow Steps

Flow chart shown in Figure 8-12 show the flow steps of the DPR flow where from step 2 to step 11 are performed using Vivado while the last step (Step No. 12) is where SDK is used to run our software "C" code on the processor.

#### 8.3.1.1 Step 1

"Prepare a Black Box Top Module and prepare each RM to have the same I/O ports."

Black Box Module is a module where you only define the input and output ports of the module without performing any logic. This module will be used in the static design (Reconfigurable Partition shown in Figure 8-10). This Black Box Module will be modified later in the following steps with the RMs (3G, LTE, Wi-Fi).

The top module of each RM must have the same module name and same input and output ports of the Black Box Module.

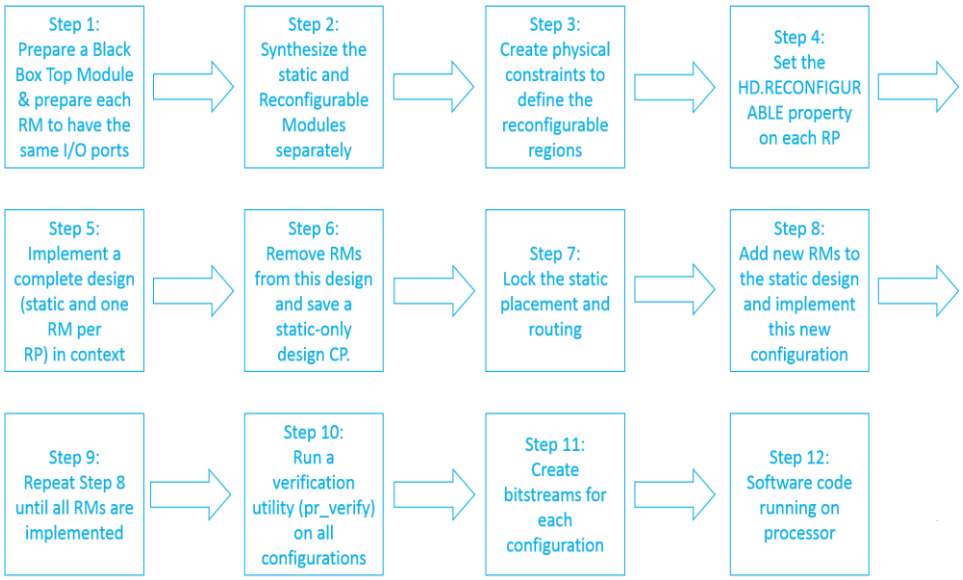


Figure 8-11: DPR flow

Example: If you have a RM originally with 3 Inputs, 5 Output and another RM originally with 6 Input, 3 Outputs. The number of input & output ports of the black box and the new top modules after modification should be maximum of each case, then we should have 6 Input ports and 5 Output Ports as shown in Table 8-2 & the top Module used is as shown in Figure 8-12.

Table 8-2: Top Module pins

| | Actual number of pins | | Modified number of pins | |
|---|---|---|---|---|
| | Input | Output | Input | Output |
| RM 1 | 3 | 5 | 6 | 5 |
| RM 2 | 6 | 3 | 6 | 5 |
| Black Box | - | - | 6 | 5 |


Figure 8-12: Black Box Module

#### 8.3.1.2 Step 2

"Synthesize the static and Reconfigurable Modules separately."

In this step the static design of the DPR system is synthesized with the black box. This will create a DCP file in <project_path>/project_name.runs/Synth. If you opened the synthesized design you will get a critical warning that the tool could not resolve a non-primitive black box cell.

Each RM (Standard Chain) is synthesized in a separate project but with synthesize settings as shown in Figure 8-13:

1- BUFG = 0, BUFG is a non-reconfigurable module

2- Added option "–mode out_of_context" to define that this synthesized block will be a part of a bigger design which is the static design in our case.

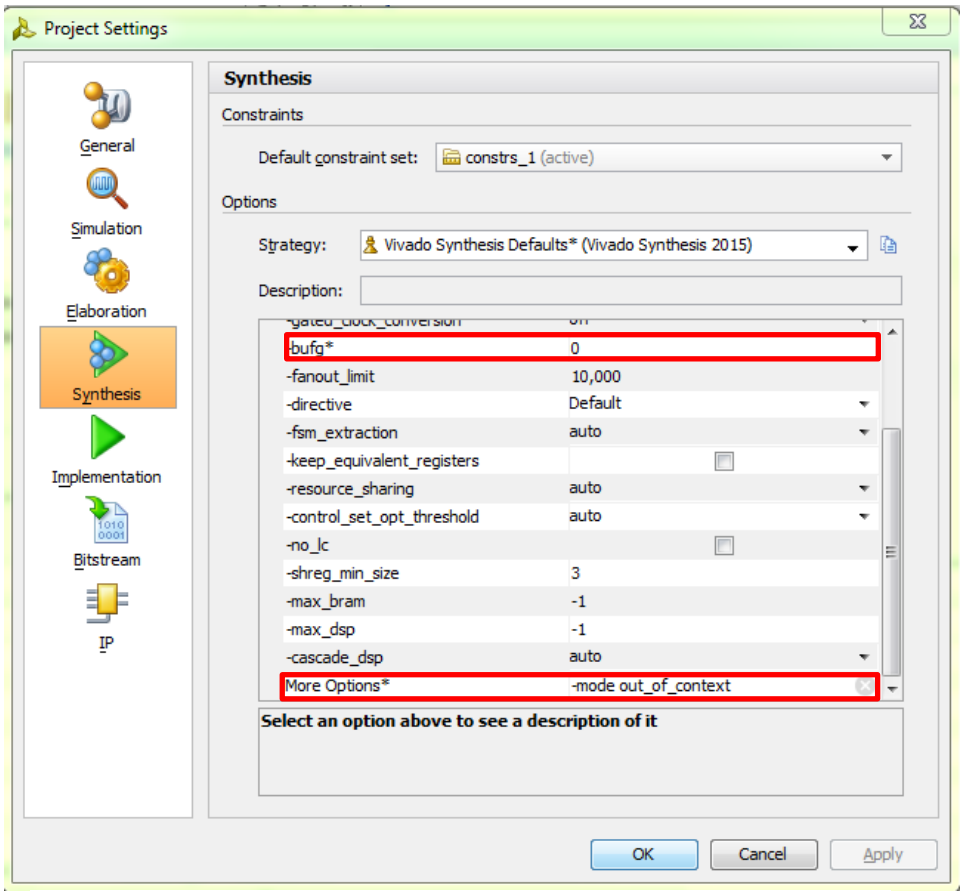


Figure 8-13: Step 2 in details

Then generating the DCP of each synthesized RM.

### 8.3.1.3 Step 3

"Create physical constraints (Pblocks) to define the reconfigurable regions."

Open the static DCP generated in step 2 to do the floor-planning of the Reconfigurable Partition, as shown in Figure 8-14, with resources which can cover the resources needed by each RM. In this case we have only one reconfigurable partition.

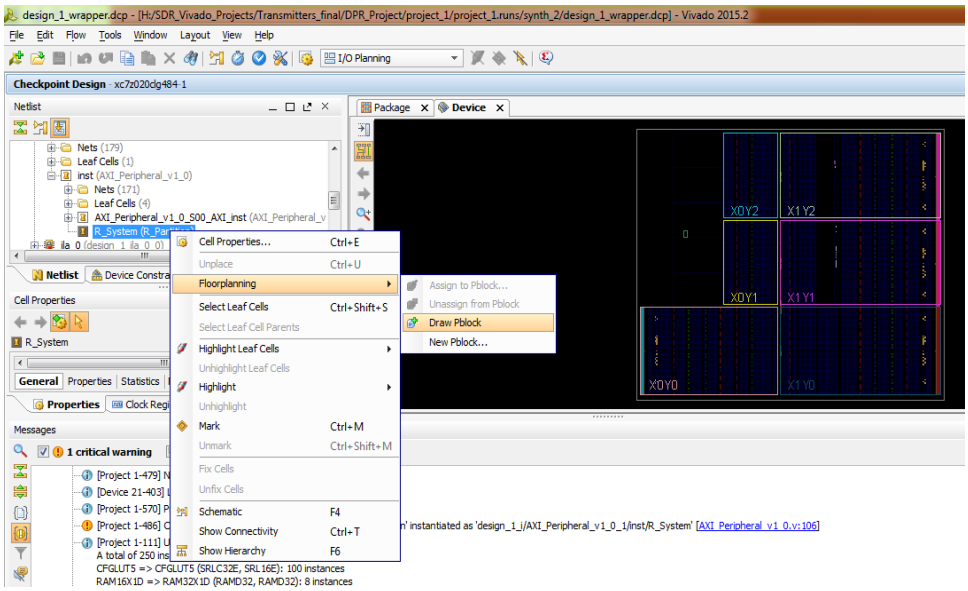


Figure 8-14: Step 3 in details

Noting that you can save this floor-planning performed using the Tcl command "write_xdc<path>/file_name" in Tcl console. So, you can use the Tcl command "read_xdc<path>/file_name" command next time instead of drawing Pblock each time. Make sure the Pblock cover all the resource required by each RM. In our case the maximum resources required goes to the LTE RM so if LTE RM fit into the Pblock it would be guaranteed that 3G & Wi-Fi RMs will fit. That would be checked in step 5. Pblock is chosen as shown in Figure 8-15.

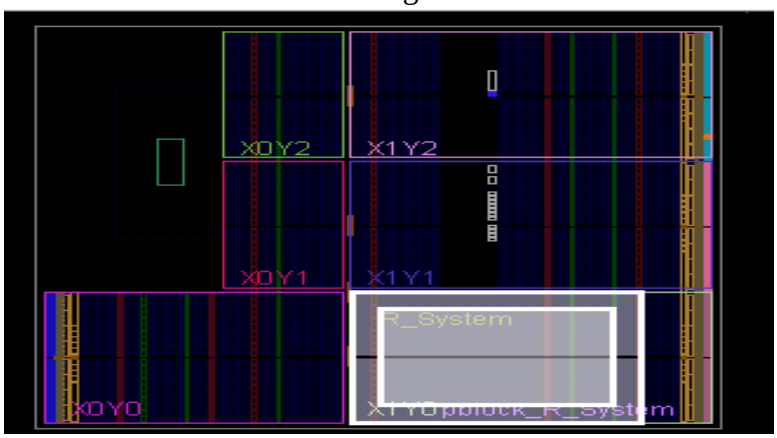


Figure 8-15: Pblock in our project

#### 8.3.1.4 Step 4

"Set the HD. RECONFIGURABLE property on each Reconfigurable Partition."

Run the Tcl command in Tcl console to define the cell which is still black box till this step "set_property HD.RECONFIGURABLE 1 [get_cells design_1_i/ AXI_Peripheral_v1_0_1/inst/R_System]", the cell is defined as reconfigurable module. Where "design_1_i/AXI_Peripheral_v1_0_1/inst/R_System" is the cell name in the design.

#### 8.3.1.5 Step 5

"Implement a complete design (static and one Reconfigurable Module per Reconfigurable Partition) in context."

Now, we read one of the RMs DCP generated in Step 2 in the Black Box cell. We start with LTE just to make sure that Pblock cover all the resources needed by LTE RM.

Tcl command: read_checkpoint -cell cell_name <4g_DCP_Path>/4g_synth.dcp

Check Pblock properties to guarantee that all required source by LTE RM are available as shown in Figure 8-16.

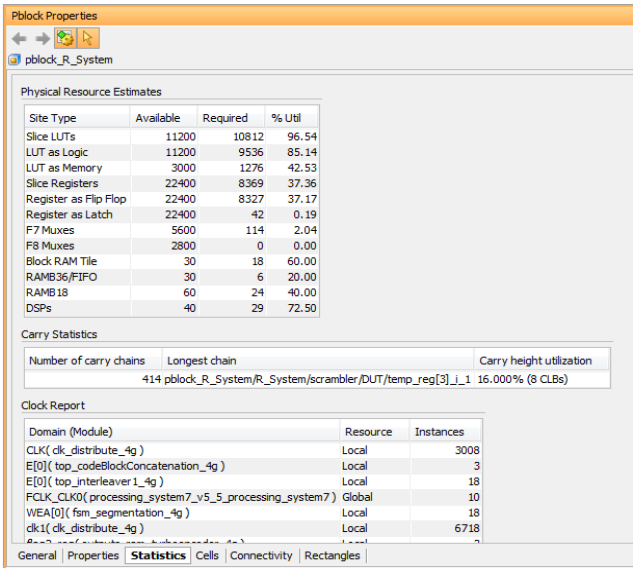


Figure 8-16: Pblock properties

Then Implement Design using the Three Tcl command "opt_design ,place_design route_design" and Write the DCP of the implemented design as it will be used in the generation of bitstreams in Step 12.

#### 8.3.1.6 Step 6

"Remove Reconfigurable Modules from this design and save a static-only design checkpoint."

Remove LTE block from the Pblock to make it Black Box again by using the Tcl command: "update_design -cell design_1_i/ AXI_Peripheral_v1_0_1/ inst/ R_System -black_box ".

Then write the DCP as this DCP represents the fully implemented and routed static design that would be used later in the implementation of other RMs.

#### 8.3.1.7 Step 7

"Lock the static placement and routing."

This step is not to change the static design routed and implemented during the implementation of other RMs.

#### 8.3.1.8 Step 8

"Add new Reconfigurable Modules to the static design and implement this new configuration."

Read one of the other RMs DCP generated in Step 2 (3G or Wi-Fi) in the Black Box cell. Implement the design and write the DCP of the implemented design as in Step No. 5 of LTE.

#### 8.3.1.9 Step 9

"Repeat Step 8 until all Reconfigurable Modules are implemented."

Update the cell to be black box again after 3G or Wi-Fi is implemented and implement the other one.

#### 8.3.1.10 Step 10

"Run a verification utility (pr_verify) on all configurations."

Verify that the all the 3 implemented DCP are compatible in Step No. 5, 8 & 9 using the Tcl command: "pr_verify -initial <path>/4G_Implementation.dcp -additional {<path>/3G_Implementation.dcp <path>/WiFi_Implementation.dcp }"

#### 8.3.1.11 Step 11

"Create bitstreams for each configuration."

Create Full and Partial bitstreams.

#### 8.3.1.12 Step 12

"Software code running on processor."

Figure 8-4 shows the architecture of the Design and how the PL & PS in our design are communicating where the DPR System shown in the Figure is as shown in Figure 8-10. This architecture simplifies the understanding of the flow chart describing the code used show in Figure 8-17 to run on the processor.

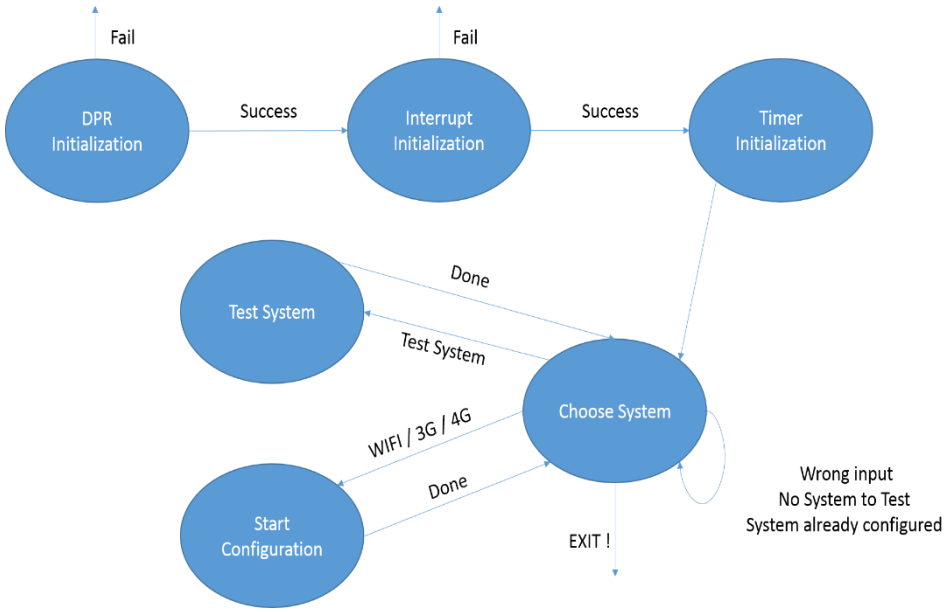


Figure 8-17: DPR flow chart

### 8.3.2 DPR Results

The Generated bitstream for the reconfigurable partition in 8.3.1 is 617 KB which mean it is around 5 Mbits.

All the gotten results is shown in Table 8-3 showing the configuration time & the power.

Table 8-3: DPR results

| | Power(mW) | Reconfiguration Time(ms) | | |
|---|---|---|---|---|
| | | HWICAP | PRC | Ideal |
| WIFI | 20.47 | 300 | 5.4 | 1.6 |
| 3G | 1.15 | | | |
| 4G | 148.875 | | | |
| | Average Power = 71.245 | | | |

CHAPTER 9

# Results, Conclusion & Future Work

In this chapter, we are going to conclude our achievements and results through a year full of team work, enthusiasm, hard work and research, then we are showing the conclusion of the performed work and the future work which should be done next. The future work needed is in three main tracks. The first one is to improve the reconfiguration between the different systems. The second one is to let the transmitters and receivers communicate with each other through the air interface using two USRPs. The third track is to add more systems such as GSM and Bluetooth.

## 9.1 Results

Our project is an experience of both hardware and software skills. And in our project we have been keen on verifying our results to make sure of the success of our work.

The final results of our work

- Optimization of 3G transmitter HDL codes & HDL and MATLAB implementation of 3G Receiver.
- Optimization of Wi-Fi transmitter HDL codes & HDL and MATLAB implementation of Wi-Fi Receiver.
- Optimization of LTE transmitter HDL codes, MATLAB implementation of LTE Receiver & HDL implementation of most of the LTE Receiver blocks.
- HDL and MATLAB implementation of 2G transmitter and receiver.
- Building testing environment on FPGA to test all the implemented chains.
- Building DPR system using two different controllers of ICAP (HWICAP & PRC).

## 9.2 Conclusion

By using the PDR to switch between the three systems this reduces the dissipated power in a large way. In current mobile phones, each system is implemented as a separate system which is always running even if it's not activated. Using the PDR here decreases the dissipated power as only one system is active and running at each time.

Another advantage for using the SDR is that it can be easily tuned from one system to another with little switching time. As a result, it is clear that using SDRs in mobiles has many advantages over the conventional mobiles used now. Right now the future is moving towards this concept.

## 9.3 Future Work

### 9.3.1 PDR future work

The future work in the Partial Dynamic Reconfiguration (PDR) should be directed to two different paths:

- Decrease the reconfiguration time between the three transmitters.
- Develop the test environment required to perform the PDR on the three receivers.

#### 9.3.1.1 Decrease reconfiguration time between transmitters

As mentioned before in Chapter 8, the reconfiguration time between the three transmitters is around 10 milliseconds. This reconfiguration time can be reduced by:

- Using multiple partitions instead of single partitions.
- Using different reconfiguration techniques such as
    1. ZYCAP, Open source controller of ICAP.
    2. Implementing our specific controller for SDR to reach maximum throughput.

#### 9.3.1.2 PDR between receivers

PDR was performed successfully with the three transmitters. The next step is to go on with the same approach to perform the reconfiguration between the three

receivers. The testing environment should be developed to switch between the receivers. Once this is achieved, the full communication between the transmitters and receivers can be achieved on the same FPGA.

### 9.3.2 Communicating through air

The previous work was to simulate communication systems in baseband only but without using real antennas to send and receive data. Only noise was added to the transmitted data just to simply model the real case. As shown in **Error! Reference source not found.** which contains the simple communication system, the completed part is DSP part. The future work should include working in the other modules other than the DSP one.

The transmitter and receiver should be communicating through air such that the whole communication system is complete. This requires one FPGA for the transmitter, one FPGA for the receiver, one USRP for the transmitter, and one USRP for the receiver.

### 9.3.3 Adding new systems

It is clear that there are four implemented and tested systems. These four systems are 3G, 4G, Wi-Fi & 2G. Other systems can be added to the existing systems. One of them can be the Bluetooth system.

# References


[1] I. F. Akyildiz, W.-Y. Lee, M. C. Vuran, and S. Mohanty, "NeXt generation/ dynamic spectrum access /cognitive radio wireless networks: a survey," *Computer Networks*, vol. 50, pp. 2127–2159, 2006.

[2] T. Yücek and H. Arslan, "A survey of spectrum sensing algorithms for cognitive radio applications," *Communications Surveys & Tutorials, IEEE*, vol. 11,pp. 116–130, 2009.

[3] A. A. Bletsas, *Intelligent antenna sharing in cooperative diversity wireless networks*. PhD thesis, Citeseer, 2005.

[4] J. Mitola III and G. Q. Maguire Jr, "Cognitive radio: making software radios more personal," *Personal Communications*, IEEE, vol. 6, pp. 13–18, 1999.

[5] J. Mitola, "Cognitive Radio—An Integrated Agent Architecture for Software Defined Radio," 2000.

[6] J. Mitola, "Cognitive Radio—A Real Implementation for Software Defined Radio," 2002.

[7] 3GPP, Physical channels and mapping of transport channels onto physical channels (FDD) (Release 12), 2014.

[8] Ahmed Hamdy, Ahmed Khaled, Karim Mohammed, Yahia Ramadan, Yahia Zakaria, HSPA+ uplink physical layer simulation modelling and hardware impelementation, Cairo : Cairo University, Faculty of Engineering, 2011

[9] 3GPP, Multiplexing and channel coding (FDD) (Release 12), 2014.

[10] IEEE, IEEE 802.11-2012, Part 11: Wireless LAN Medium Access Control,2012.

[11] Asmaa Rayan, Alaa Othman, Maha Abd El-Maqsoud, OFDM over optical fiber channel, Cairo: Cairo University, Faculty of Engineering, 2015.

[12] Architecture of FPGAs and CPLDs: A Tutorial Stephen Brown and Jonathan Rose Department of Electrical and Computer Engineering University of Toronto.

[13] Programming Technologies for Field Programmable Gate Arrays Tanvir ahmed abbasi and mohammad Usaid abbasi University Polytechnic, Department of Electronics and Communication, Faculty of Engineering and Technology, Jamia Millia Islamia, New Delhi, India.

[14] MicroBlaze Processor Reference Guide Embedded Development Kit EDK 10.1i.

[15] Zynq-7000 All Programmable SoC Technical Reference Manual.

[16] ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC User Guide.

[17] 7 Series FPGAs Configurable Logic Block User Guide.

[18] A Testing Environment for Multi-Clock Systems on Xilinx ZynQ SoC

[19] Xilinx Inc. “AXI Reference Guide UG761”, March 7, 2011.

[20] Xilinx Inc. “AXI DMA PG021”, March 7, 2011.

[21] Xilinx Inc. “ZC702 Evaluation Board reference manual UG585”, September 2015.

[22] Xilinx Inc. “AXI HWICAP PG134” v3.0, October 2016.

[23] Xilinx Inc. “Partial Reconfiguration Controller PG193” v1.1, October 2016.

[24] Ahmed Kamaleldin, Ahmed Nagy and Hassan Mostafa, " Design Guidelines for the High-Speed Dynamic Partial Reconfiguration Based Software Defined Radio Implementations on Xilinx ZynQ FPGA,".